The Mnemosyne architecture defines a compatible framework for a family of implementations with a range of capabilities. The following implementation-defined parameters are used in the rest of the document in boldface. The value indicated is for MicroUnity's first Mnemosyne implementation.

Parameter	Interpretation	Value	Range of legal values
C	$\log_2$ logical memory words in SRAM cache	13	$\mathbf{C} \geq 1$
B	$\log_2$ physical memory words in SRAM cache physical memory block	11	$\mathbf{B} \geq 1$
S	number of bits per word of an SRAM physical memory block	9	$\mathbf{S} > 0$
t	size of tag field in cache entry, in bits	13	$\mathbf{t} = 2\mathbf{P} + \mathbf{E} - \mathbf{C}$
e	size of ECC field in cache entry, in bits	8	$\mathbf{e} \geq \log_2(8\mathbf{W}+\mathbf{t}+1+\mathbf{e})+1$
n	number of physical memory blocks used to produce a logical memory word	10	$\mathbf{n} \geq \frac{8\mathbf{W}+\mathbf{t}+1+\mathbf{e}}{\mathbf{S}}$
N	number of SRAM physical memory blocks, not including redundant blocks	40	$\mathbf{N} = \mathbf{n}(2^{\mathbf{C}-\mathbf{B}})$
D	number of divisions of SRAM physical memory blocks covered by separate sets of redundant blocks	2	$1 \leq \mathbf{D} \leq 16$
R	number of redundant SRAM physical memory blocks in each redundancy division	2	$1 \leq \mathbf{R} \leq 16$
P	number of DRAM row and column address interface pins	12	$9 < \mathbf{P} < (\mathbf{A}*8-\mathbf{E})/2$
K	number of address interface pins which may be configured as row-address-only pins	0	$0 \leq \mathbf{K} \leq \mathbf{P}$
I	$\log_2$ of number of interleaved accesses in DRAM interface	2	$0 < \mathbf{I} < 16$
E	$\log_2$ of number of banks of DRAM expansion	2	$\mathbf{I} \leq \mathbf{E} \leq 15$

MU 0023411

Interfaces and Block Diagram

Mnemosyne uses two Hermes unidirectional, byte-wide, differential, packet-oriented data channels for its main, high-bandwidth interface between a memory control unit and Mnemosyne's memory. This interface is designed to be cascadeable, with the output of a Mnemosyne chip connected to the input of another, to expand the size of memory that can be reached via a single set of data

channels. An external memory control unit is in complete control of the selection and timing of operations within Mnemosyne and in complete control of the timing and content of information on the high-bandwidth interfaces.

A Cerberus bit-serial interface provides access to configuration, diagnostic and tester information, using TTL signal levels at a moderate data rate.

Mnemosyne contains additional interfaces to conventional dynamic random-access memory devices (DRAM) using TTL signals. Each Mnemosyne device contains output signals to independently control four banks of DRAM memory; each bank is nominally 9 bytes wide, and connects to a single set of bidirectional data interface pins. Each DRAM bank may use 24-bit addresses, to handle up to 16M-"word" DRAM memory capacity (such as 16Mx4 organized, 64-Mbit DRAM). Up to four banks of DRAM may be connected to each Mnemosyne device, permitting up to 0.5 Gbyte of DRAM per Mnemosyne chip.

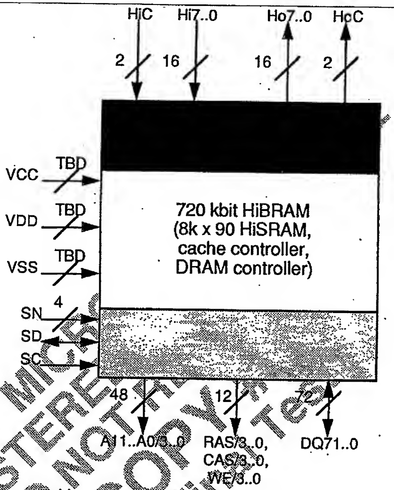
Nearly all Mnemosyne circuits use a single power supply voltage, nominally at 3.3 Volts (5% tolerance). A second voltage of 5.0 Volts (5% tolerance) is used only for TTL interface circuits. Power dissipation is TBD. Initial packaging is TAB (Tape Automated Bonding).

Pin assignments are to be defined; there are 174 signal pins and 466 pins for 3.3V power, 5.0V power and substrate, for a total of 640 pins.

count	pin	meaning
18	HiC, $Hi_{7..0}$	hi-bandwidth input
18	HoC, $Ho_{7..0}$	hi-bandwidth output
72	$DQ_{71..0}$	DRAM data
48	$A_{11..0 \, 3..0}$	DRAM address
12	$RAS_{3..0}$, $CAS_{3..0}$, $WE_{3..0}$	DRAM control
6	SC, SD, $SN_{3..0}$	Cerberus interface
174		total signal pins
?	VDD	3.3 V above VSS
?	VCC[44]	5.0 V above VSS
?	VSS	most negative supply
640		total pins

[44] Internal circuit documentation names this signal VDDO.

The following is a diagram of the Mnemosyne device interfaces: (Numerical values are shown for MicroUnity's first implementation.)



Mnemosyne external block diagram

Absolute Maximum Ratings	MIN	NOM	MAX	UNIT

Recommended operating conditions	MIN	NOM	MAX	UNIT	REF
V_T: Termination equivalent voltage	4.5	5.0	5.5	V	
Main supply voltage VDD	3.14	3.3	3.47	V	VSS
TTL supply voltage VCC	4.75	5.0	5.25	V	VSS
Operating free-air temperature	0		70	C	

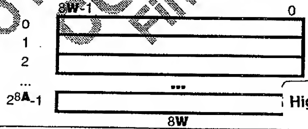
Electrical characteristics	MIN	TYP	MAX	UNIT	REF
V_{OH}: H-state output voltage HoC, $Ho_{7..0}$				V	VDD
V_{OL}: L-state output voltage HoC, $Ho_{7..0}$				V	VDD
V_{IH}: H-state input voltage HiC, $Hi_{7..0}$				V	VDD
V_{IL}: L-state input voltage HiC, $Hi_{7..0}$				V	VDD
I_{OH}: H-state output current HoC, $Ho_{7..0}$				mA	
I_{OL}: L-state output current HoC, $Ho_{7..0}$				mA	
I_{IH}: H-state input current HiC, $Hi_{7..0}$				mA	
I_{IL}: L-state input current HiC, $Hi_{7..0}$				mA	
C_{IN}: Input capacitance HiC, $Hi_{7..0}$				pF	
C_{OUT}: Output capacitance HoC, $Ho_{7..0}$				pF	
V_{OH}: H-state output voltage $A_{11..0}{}_{3..0}$, $RAS_{3..0}$, $CAS_{3..0}$, $WE_{3..0}$, $DQ_{71..0}$	2.4		5.5	V	VSS
V_{OL}: L-state output voltage $A_{11..0}{}_{3..0}$, $RAS_{3..0}$, $CAS_{3..0}$, $WE_{3..0}$, $DQ_{71..0}$	0		0.4	V	VSS
V_{OL}: L-state output voltage SD	0		0.4	V	VSS
V_{IH}: H-state input voltage $DQ_{71..0}$	2.4		5.5	V	VSS
V_{IL}: L-state input voltage $DQ_{71..0}$	-0.5		0.8	V	VSS
V_{IH}: H-state input voltage SD	2.0		5.5	V	VSS
V_{IH}: H-state input voltage SC, $SN_{3..0}$	2.0		5.5	V	VSS
V_{IL}: L-state input voltage SC, SD, $SN_{3..0}$	-0.5		0.8	V	VSS
I_{OH}: H-state output current $A_{11..0}{}_{3..0}$, $RAS_{3..0}$, $CAS_{3..0}$, $WE_{3..0}$, $DQ_{71..0}$				µA	
I_{OL}: L-state output current $A_{11..0}{}_{3..0}$, $RAS_{3..0}$, $CAS_{3..0}$, $WE_{3..0}$, $DQ_{71..0}$			16	mA	
I_{OL}: L-state output current SD			16	mA	
I_{OZ}: Off-state output current SD	-10		10	µA	
I_{OZ}: Off-state output current $DQ_{71..0}$	-10		10	µA	
I_{IH}: H-state input current SC, $SN_{3..0}$	-10		10	µA	
I_{IL}: L-state input current SC, $SN_{3..0}$	-10		10	µA	
C_{IN}: Input capacitance SC, $SN_{3..0}$			4.0	pF	
C_{OUT}: Output or input-output capacitance, SD, $A_{11..0}{}_{3..0}$, $RAS_{3..0}$, $CAS_{3..0}$, $WE_{3..0}$, $DQ_{71..0}$			4.0	pF	

Switching characteristics	MIN	TYP	MAX	UNIT
t_{BC}: HiC clock cycle time	1000			ps
t_{BCH}: HiC clock high time	400			ps
t_{BCL}: HiC clock low time	400			ps
t_{BT}: HiC clock transition time			100	ps
t_{BS}: set-up time, $Hi_{7..0}$ valid to HiC xition	200		100	ps
t_{BH}: hold time, HiC xition to $Hi_{7..0}$ invalid	-200		-100	ps
t_{OS}: skew between HoC and $Ho_{7..0}$	-50		50	ps
t_C: SC clock cycle time	50			ns
t_{CH}: SC clock high time	20			ns
t_{CL}: SC clock low time	20			ns
t_T: SC clock transition time			5	ns
t_S: set-up time, SD valid to SC rise				ns
t_H: hold time, SC rise to SD invalid				ns
t_{OD}: SC rise to SD valid	5			ns

Logical and Physical Memory Structure

Mnemosyne defines two regions: a memory region, implemented by an on-device static RAM memory cache backed by standard DRAM memory devices, and a configuration region, implemented by on-device read-only and read/write registers. These regions are accessed by separate interfaces: the Hermes channel used to access the memory region, and the Cerberus serial interface used to access the configuration region. These regions are kept logically separate.

The Mnemosyne logical memory region is an array of 2^{8A} words of size W bytes. Each memory access, either a read or write, references all bytes of a single block. All addresses are block addresses, referencing the entire block.



Logical memory organization

Mnemosyne's DRAM memory physically consists of one or more banks of multiplexed-address DRAM memory devices. A DRAM bank consists of a set of DRAM devices which have the corresponding address and control signals connected together, providing one word of W bytes of data plus ECC information with each DRAM access.

Mnemosyne's SRAM memory is a write-back (write-in) single-set (direct-mapped) cache for data originally contained in the DRAM memory. All accesses to

Mnemosyne memory space maintain consistency between the contents of the cache and the contents of the DRAM memory.

Mnemosyne's configuration region consists of read-only and read/write registers. The size of a logical block in the configuration memory space is eight bytes: one octlet.

Communications Channels

High-bandwidth

Mnemosyne uses the Hermes high-bandwidth channel and protocols, implementing a slave device.

Mnemosyne operates two Hermes high-bandwidth communications channels, one input channel and one output channel.

Mnemosyne uses the Hermes packet structure. Mnemosyne's SRAM memory serves as the Hermes-designated cache, and Mnemosyne DRAM memory corresponds to the Hermes-designated device.

Configuration-region registers provide a low-level mechanism to detect skew in the byte-wide input channel, and to adjust skew in the byte-wide output channel. This mechanism may be employed by software to adaptively adjust for skew in the channels, or set to fixed patterns to account for fixed signal skew as may arise in device-to-device wiring.

Serial

A Cerberus serial bus interface is used to configure the Mnemosyne device, set diagnostic modes and read diagnostic information, and to enable the use of the part within a high-speed tester.

The serial port uses the Cerberus serial bus interface.

DRAM

The DRAM interface uses TTL levels to communicate with standard, high-capacity dynamic RAM devices. The data path of the interface is 8W+ e bits. The DRAM components used may have a maximum size of 2^{2P} words by k bits, where the minimum value of k is determined by capacitance limits. (Larger values of k, up to 8W+e, meaning fewer components are required to assemble a word of DRAMs, are always acceptable.)

Error Handling

Mnemosyne performs error handling compliant with Hermes architecture.

For the current implementation, the following errors are designed to be detected and known not detected by design:

errors detected	errors not detected
invalid check byte	invalid identification number
invalid command	internal buffer overflow
invalid address	invalid check byte on idle packet
uncorrectable error in SRAM cache	
uncorrectable error in DRAM memory	

Detection of an uncorrectable error in either the SRAM cache or the DRAM memory results in the generation of an error response packet and other actions more fully described elsewhere.

Upon receipt of the error response packet, the packet originator must read the status register of the reporting device to determine the precise nature of the error. Mnemosyne devices reporting an invalid packet will suppress the receipt of additional packets until the error is cleared, by clearing the status register. However, such devices may continue to process packets which have already been received, and generate responses. Upon taking appropriate corrective actions and clearing the error, the packet originator should then re-send any unacknowledged commands.

Because of the large difference in clock rate between the high-bandwidth Hermes channel and the Cerberus serial bus interface, it is generally safe to assume that, after detecting an error response packet, an attempt to read the status register via Cerberus will result in reading stable, quiescent error conditions and that the queue of outstanding requests will have drained. After clearing the status register via Cerberus, the packet originator may immediately resume sending requests to the Mnemosyne device.

Cerberus Registers

Mnemosyne's configuration registers comply with the Cerberus and Hermes specifications. Configuration registers are internal read/only and read/write registers which provide an implementation-independent mechanism to query and control the configuration of a Mnemosyne device. By the use of these registers, a user of a Mnemosyne device may tailor the use of the facilities in a general-purpose implementation for maximum performance and utility. Conversely, a supplier of a Mnemosyne device may modify facilities in the device without compromising compatibility with earlier implementations.

Read/only registers supply information about the Mnemosyne implementation in a standard, implementation-independent fashion. A Mnemosyne user may take advantage of this information, either to verify that a compatible implementation of Mnemosyne is installed, or to tailor the use of the part to conform to the characteristics of the implementation. The read/only registers occupy addresses 0..5. An attempt to write these registers may cause a normal or an error response.

REDACTED

Read/write registers select the mapping of addresses to SRAM and DRAM banks, control the internal SRAM and DRAM timing generators, and select power and voltage levels for gates and signals. The read/write registers occupy addresses 6..11, 16..19, and 32.

Reserved registers in the range 12..15, 20..31, and 33..63 must appear to be read/only registers with a zero value. An attempt to write these registers may cause a normal or an error response.

Reserved registers in the range 64..2^{16}-1 may be implemented either as read/only registers with a zero value, or as addresses which cause an error response if reads or writes are attempted.

The format of the registers is described in the table below. The **octlet** is the Cerberus address of the register; **bits** indicate the position of the field in a register. The **value** indicated is the hard-wired value in the register for a read/only register, and is the value to which the register is initialized upon a reset for a read/write register. If a reset does not initialize the field to a value, or if initialization is not required by this specification, a * is placed in or appended to the value field. The **range** is the set of legal values to which a read/write register may be set. The **interpretation** is a brief description of the meaning or utility of the register field; a more comprehensive description follows this table.

octlet	bits	field name	value	range	interpretation
0	63..16	**architecture code**	0x00 40 a3 49 d2 e4		Identifies memory device as compliant with MicroUnity Mnemosyne architecture.
	15..0	**architecture revision**	0x01 00		Device complies with architecture version 1.0.

octlet	bits	field name	value	range	interpretation
1	63..16	**implementor code**	0x00 40 a3 24 6d f3		Identifies Mnemosyne Memory device as implemented by MicroUnity.
	15..0	**implementor revision**	0x01 00		Implementation version 1.0.

octlet	bits	field name	value	range	interpretation
2	63..16	**manufacturer code**	0x00 40 a3 92 b6 79		Identifies initial manufacturer of Mnemosyne Memory device implemented by MicroUnity.
	15..0	**manufacturer revision**	0x01 00		Manufacturing version 1.0.

octlet	bits	field name	value	range	interpretation
3	63..16	**serial number**	0		This device has no serial number capability.
		dynamic address	0		This device has no dynamic addressing capability.

octlet	bits	field name	value	range	interpretation
4	63..60	**A**	4	0..15	size of a Mnemosyne address
	59..56	**$\log_2$W**	3	0..15	size of a Mnemosyne word
	55..48	**C**	13	0..25 5	$\log_2$ of cache capacity in words
	47..40	**N**	40	0..25 5	number of cache sub-blocks (excluding redundant blocks)
	39..36	**D**	2	0..15	Number of divisions of cache-blocks covered by separate sets of redundant blocks. A zero value signifies 16 divisions.
	36..32	**R**	2	0..15	Number of redundant blocks per division. A zero value signifies 16 redundant blocks.
	31..28	**P**	12	0..15	Number of row and column address interface pins
	27..24	**K**	0	0..15	Maximum value by which column address pin count may be less than row address pin count.
	23..20	**E**	2	0..15	$\log_2$ of number of banks of DRAM expansion
	19..16	**I**	2	0..15	$\log_2$ of maximum interleaving level in DRAM interface.
	15..0		0		Reserved for definition in later revision of Mnemosyne architecture

octlet	bits	field name	value	range	interpretation
5	63.0		0		Reserved for definition in later revision of Mnemosyne architecture

octlet	bits	field name	value range		interpretation
6	63	**reset**	1	0..1	set to invoke device's circuit reset
	62	**clear**	1	0..1	set to invoke device's logic clear
	61	**selftest**	0	0..1	set to invoke device's selftest: bits 60..48 may indicate depth of selftest
	60	**tester**	0	0..1	set to invoke tester mode
	59	**isolate/ synch**	0	0..1	~tester mode: if set, suppress cache misses/writebacks. tester mode: synch up
	58	**source**	0	0..1	~tester mode: set to 0 tester mode: source/analyzer
	57	**ECC disable**	0	0..1	disable ECC checking: can be set during normal operating mode
	56..50	**0**	0	0	Reserved for additional mode bits
	49..48	**module id**	0	0..3	Module identifier.
	47	**PLL bypass**	0	0..1	Setting this bit causes the PLL to be bypassed; the input clock signal is used directly.
	46..45	**PLL range extension**	0	0	Reserved for extensions to the PLL range control field.
	44	**PLL range**	0	0..1	Set to 0 if the PLL is operating at a low frequency; 1 if the PLL is operating at a high frequency.
	43..40	**output slope control**	0	0..15	Output slope for DRAM control signals
	39..36	**output slope address**	0	0..15	Output slope for DRAM address signals
	35..32	**output slope data**	0	0..15	Output slope for DRAM data signals
	31..29	**SRAM timing extension**	0	0	Reserved for additional SRAM timing control bits.
	28	**SRAM timing**	0	0..1	Set to 1 to extend SRAM timing by one clock cycle.
	27..24	**ECC seed extension**	0	0	extend ECC seed value when **W** > 8
	23..16	**ECC seed**	0	0..255	Value to modify ECC code computed on incoming data. Used to exercise ECC detection/correction logic, or to write arbitrary patterns into memory.
	15..8	**cidle 0**	0	0..255	Value transmitted on idle Hermes output channel when output clock zero (0).
	7..0	**cidle 1**	255	0..255	Value transmitted on idle Hermes output channel when output clock one (1).

octlet	bits	field name		value range	interpretation
7	63	**reset/clear/ selftest complete**	1	0..1	This bit is set when a reset, clear or selftest operation has been completed.
	62	**reset/clear/ selftest status**	1	0..1	This bit is set when a reset, clear or selftest operation has been completed successfully.
	61	**check byte error**	0	0..1	This bit is set when a received input packet has an incorrect check byte.
	60	**address error**	0	0..1	This bit is set when a received input request has an address not present on the device as configured.
	59	**command error**	0	0..1	This bit is set when a packet is received on the Hermes input channel with an improper command.
	58	**un-correctable ECC error**	0	0..1	This bit is set when an uncorrectable error is discovered in memory.
	57	**correctable ECC error**	0	0..1	This bit is set when a correctable error is discovered in memory.
	56	**other error**	0	0..1	This bit is set when other errors not otherwise specified occur.
	55..53	**0**	0	0	Reserved
	52..48	**PMOS drive strength**		0..15	This read/only field indicates the drive strength of PMOS devices expressed as a digital binary value.
	47..41	**0**	0	0	Reserved
	40	**PLL in range**		0..1	This bit indicates that the Hermes input channel clock and the PLL are at rates such that the PLL can lock.
	39..30	**0**	0	0	Reserved
	29	**ECC location flag**	0	0..1	0 if ECC error was in cache memory, 1 if ECC error was in DRAM memory.
	28	**dirty flag**	0	0..1	Dirty bit if error was in cache memory
	27..24	**ECC syndrome extension**	0	0	extend ECC syndrome value when **e** > 8
	23..16	**ECC syndrome**	0	0..255	Value of syndrome encountered on previous correctable or uncorrectable ECC error.
	15..8	**raw 0**	0	0..255	Value sampled on Hermes input channel when input clock is zero (0).
	7..0	**raw 1**	255	0..255	Value sampled on Hermes input channel immediately following sample value in **raw 0** register.

octlet	bits	field name	value range		interpretation
8	63..32	**0**	0		Reserved for handling larger address spaces.
	31..0	**ECC addr**	0	$0..2^{32}-1$	Address at which an ECC error was detected.

octlet	bits	field name	value range		interpretation
9	63..60	**log_2id**	0	0..**I**	Number of DRAM interleaving levels can be computed as **id** = $2^{\mathbf{log2id}}$.
	59..56	**expand**	0	1..**E**	Number of DRAM banks.
	55..52	**r**	0	9..**P**	Number of bits in DRAM row address
	51..48	**c**	0	9..**P**	Number of bits in DRAM column address
	47..40	**t1**	0	0..15	Address set up time relative to RAS
	39..32	**t2**	0	0..15	Address hold time after RAS
	31..24	**t3**	0	0..15	Address set up time relative to CAS
	23..16	**t4**	0	0..15	CAS pulse width
	15..8	**t5**	0	0..15	Page mode cycle time is t3+t4+t5, Page mode CAS precharge is t3+t5
	7..0	**t6**	0	0..15	RAS precharge is t6+t1

octlet	bits	field name	value range		interpretation
10	63..56	**t7**	0	0..15	CAS to RAS set up for refresh cycle. t7 >=t1 to ensure RAS precharge is met.
	55..48	**t8**	0	0..15	Time data bus occupied from end of CAS low
	47..40	**t9**	0	0..15	Time output data on bus from start of t3
	39..32	**t10**	0	0..15	Interval between two address bus transitions
	31	**refresh enable**	0	0..1	If set, generate refresh cycles.
	30..24	**t11**	0	0..127	Interval between refresh cycles.
	23..0	**0**	0	0	Reserved

octlet	bits	field name	value range		interpretation
11..15	63..0	**0**	0	0	Reserved

Highly Confidential MU 0023422

octlet	bits	field name	value	range	interpretation
16	63..56	**process control**	0x42	0..255	Set global power and voltage swing levels.
	55..48	**IO control**	0xc2	0..255	Set power and voltage swing levels in I/O circuits.
	47..40	**clock dist 1**	0xc2	0..255	Set power and voltage swing levels in clock distribution circuits.
	39..32	**clock dist 2**	0xc2	0..255	Set power and voltage swing levels in clock distribution circuits.
	31..26	**0**	0	0	Reserved
	25..24	**digital skew clk**	0	0..3	Set number of skew delay circuits to insert in output HoC.
	23..22	**digital skew bit 7**	0	0..3	Set number of skew delay circuits to insert in output Ho7.
	21..20	**digital skew bit 6**	0	0..3	Set number of skew delay circuits to insert in output Ho6.
	19..18	**digital skew bit 5**	0	0..3	Set number of skew delay circuits to insert in output Ho5.
	17..16	**digital skew bit 4**	0	0..3	Set number of skew delay circuits to insert in output Ho4.
	15..14	**digital skew bit 3**	0	0..3	Set number of skew delay circuits to insert in output Ho3.
	13..12	**digital skew bit 2**	0	0..3	Set number of skew delay circuits to insert in output Ho2.
	11..10	**digital skew bit 1**	0	0..3	Set number of skew delay circuits to insert in output Ho1.
	9..8	**digital skew bit 0**	0	0..3	Set number of skew delay circuits to insert in output Ho0.
	7..0	**analog skew clk**	0xc2	0..255	Set power and voltage swing levels in HoC skew delay circuits.

REDACTED

octlet	bits	field name	value range		interpretation
17	63..56	**analog skew bit 7**	0xc2	0..255	Set power and voltage swing levels in Ho7 skew delay circuits.
	55..48	**analog skew bit 6**	0xc2	0..255	Set power and voltage swing levels in Ho6 skew delay circuits.
	47..40	**analog skew bit 5**	0xc2	0..255	Set power and voltage swing levels in Ho5 skew delay circuits.
	39..32	**analog skew bit 4**	0xc2	0..255	Set power and voltage swing levels in Ho4 skew delay circuits.
	31..24	**analog skew bit 3**	0xc2	0..255	Set power and voltage swing levels in Ho3 skew delay circuits.
	23..16	**analog skew bit 2**	0xc2	0..255	Set power and voltage swing levels in Ho2 skew delay circuits.
	15..8	**analog skew bit 1**	0xc2	0..255	Set power and voltage swing levels in Ho1 skew delay circuits.
	7..0	**analog skew bit 0**	0xc2	0..255	Set power and voltage swing levels in Ho0 skew delay circuits.

octlet	bits	field name	value range		interpretation
18	63..56	**SRAM pipe**	0xc2	0..255	Set power and voltage swing levels in SRAM pipeline circuits.
	55..48	**DRAM data**	0xc2	0..255	Set power and voltage swing levels in DRAM data circuits.
	47..40	**DRAM address**	0xc2	0..255	Set power and voltage swing levels in DRAM address circuits.
	39..32	**PLL in range indicator**	0xc2	0..255	Set power and voltage swing levels in PLL in-range detector circuits.
	31..24	**PLL phase detector**	0xc2	0..255	Set power and voltage swing levels in PLL phase detector circuits.
	23..16	**forward logic**	0xc2	0..255	Set power and voltage swing levels in packet forwarding logic circuits.
	15..8	**forward PLA**	0xc2	0..255	Set power and voltage swing levels in packet forwarding PLA.
	7..0	**tester logic**	0xc2	0..255	Set power and voltage swing levels in tester logic circuits.

Highly Confidential

MU 0023424

octlet	bits	field name	value	range	interpretation
19	63..56	**tester PLA**	0xc2	0..255	Set power and voltage swing levels in tester PLA.
	55..48	**dual port RAMs**	0xc2	0..255	Set power and voltage swing levels in 2-port RAM circuits.
	47..40	**big PLA**	0xc2	0..255	Set power and voltage swing levels in big PLAs.
	39..32	**small PLA**	0xc2	0..255	Set power and voltage swing levels in small PLAs.
	31..24	**pipeline interface**	0xc2	0..255	Set power and voltage swing levels in pipeline interface circuits.
	23..16	**other logic 2**	0xc2	0..255	Set power and voltage swing levels in other logic circuits.
	15..8	**other logic 1**	0xc2	0..255	Set power and voltage swing levels in other logic circuits.
	7..0	**other logic 0**	0xc2	0..255	Set power and voltage swing levels in other logic circuits.

octlet	bits	field name	value	range	interpretation
20..31	63..0	**0**	0		Reserved.

octlet	bits	field name	value	range	interpretation
32	63..56	**redundant 0**	0	0..255	Enable and address for redundant block 0 (partition 0)
	55..48	**redundant 1**	0	0..255	Enable and address for redundant block 1 (partition 0)
	47..40	**redundant 2**	0	0..255	Enable and address for redundant block 0 (partition 1)
	39..32	**redundant 3**	0	0..255	Enable and address for redundant block 1 (partition 1)
	31..0	**0**	0	0	Reserved for use with additional redundant blocks.

octlet	bits	field name	value	range	interpretation
33..63	63..0	**0**	0	0	Reserved for use with additional redundant blocks.

octlet	bits	field name	value	range	interpretation
64..65536	63..0	**0**	0	0	Reserved for use with later revisions of the architecture.

configuration memory space

MU 0023425

Identification Registers

The identification registers in octlets 0..3 comply with the requirements of the Cerberus architecture.

Highly Confidential

REDACTED

MicroUnity's company identifier is: 0000 0000 0000 0010 1100 0101.

MicroUnity's architecture code for Mnemosyne is specified by the following table:

Internal code name	Code number
Mnemosyne	0x00 40 a3 49 d2 e4

Mnemosyne architecture revisions are specified by the following table:

Internal code name	Code number
1.0	0x01 00

MicroUnity's Mnemosyne implementor codes are specified by the following table:

Internal code name	Code number
MicroUnity	0x00 40 a3 24 6d f3

MicroUnity's Mnemosyne, as implemented by MicroUnity, uses implementation codes as specified by the following table:

Internal code name	Revision number
1.0	0x01 00

MicroUnity's Mnemosyne, as implemented by MicroUnity, uses manufacturer codes as specified by the following table:

Internal code name	Code number
Rollers	0x00 40 a3 92 b6 79

MicroUnity's Mnemosyne, as implemented by MicroUnity, and manufactured by the Rollers, uses manufacturer revisions as specified by the following table:

Internal code name	Code number
1.0	0x01 00

Architecture Description Registers

The architecture description registers in octlets 4 and 5 comply with the Cerberus and Hermes specifications and contain a machine-readable version of the architecture parameters: **A**, **W**, **C**, **N**, **D**, **R**, **P**, **K**, **E**, and **I** described in this document.

Control Register

The control register is a 64-bit register with both read and write access. It is altered only by Cerberus accesses; Mnemosyne does not alter the values written to this register.

The **reset** bit of the control register complies with the Cerberus specification and provides the ability to reset an individual Mnemosyne device in a system. Setting this bit is equivalent to a power-on reset or a broadcast Cerberus reset (low level on SD for 33 cycles) and resets configuration registers to their power-on values, which is an operating state that consumes minimal current. At the completion of the reset operation, the **reset/clear/selftest complete** bit of the status register is set, and the **reset/clear/selftest status** bit of the status register is set.

The **clear** bit of the control register complies with the Cerberus specification and provides the ability to clear the logic of an individual Mnemosyne device in a system. Setting this bit causes all internal high-bandwidth logic to be reset, as is required after reconfiguring power and swing levels. At the completion of the reset operation, the **reset/clear/selftest complete** bit of the status register is set, and the **reset/clear/selftest status** bit of the status register is set.

The **selftest** bit of the control register complies with the Cerberus specification and provides the ability to invoke a selftest on an individual Mnemosyne device in a system. However, Mnemosyne does not define a selftest mechanism at this time, so setting this bit will immediately set the **reset/clear/selftest complete** bit and the **reset/clear/selftest status** bit of the status register.

The **tester** bit of the control register provides the ability to use a Mnemosyne part as a component of a high-bandwidth test system for a Mnemosyne or other part using the high-bandwidth Hermes channel. In normal operation this bit must be cleared. When the **tester** bit is set, Mnemosyne is configured as either a signal source or signal analyzer, depending on the setting of the **source** bit of the control register. Four Mnemosyne parts are cascaded to perform the signal source or signal analyzer function. When the **isolate/synch** bit is set, a synchronization pattern is transmitted on the Hermes output channel and received on the Hermes input channel to synchronize the cascade of four Mnemosynes; the **isolate/synch** bits must be turned off starting at the end of the cascade to properly terminate the synchronization operation.

When not in tester mode, the **isolate/synch** bit of the control register is used to initialize the SRAM cache and perform functional testing of the SRAM cache. This bit must be cleared in normal operation. Setting this bit and setting the **ECC disable** bit of the control register suppresses cache misses and dirty cache line writebacks, so that the contents of the SRAM cache can be tested as if it were simple SRAM memory. A read-allocate command returns the octlet data from the SRAM cache entry that would be used to cache the requested location. the data is unconditionally returned, regardless of the contents of the tag, dirty and ECC fields of the SRAM cache entry. A read-noallocate command returns an octlet in the following format:

63	62 61	60 48	47 8	7 0
d	0	tag	undefined	ECC
1	2	13	40	8

A write-allocate command writes the octlet data, along with the dirty bit set, the tag corresponding to the requested location, and valid ECC data into the SRAM cache entry that would be used to cache the requested location. A write-noallocate command writes the octlet data, along with the dirty bit cleared, the tag corresponding to the requested location, and ECC data as if the dirty bit was set, into the SRAM cache entry that would be used to cache the requested location. The **ECC seed** field of the control register can be set to alter the ECC data that would otherwise be written to the SRAM cache entry, to write completely arbitrary patterns, or to write patterns in which the dirty bit is cleared and the ECC data is value.

The **ECC disable** bit of the control register causes Mnemosyne to ignore ECC errors in the SRAM cache and in the DRAM memory. This bit may be set during normal operation of Mnemosyne.

The **module id** field of the control register sets the module address for Mnemosyne. The module address defines which one of four module addresses Mnemosyne will select to answer to read and write requests.

Setting the **PLL bypass** bit of the control register causes the internal clocking of the high-bandwidth logic to operate off the input clock directly. This bit is cleared during normal operation.

The **PLL range** field of the control register is used to select an operating range for the internal PLL. A three-bit field is reserved for this function, of which one bit is currently defined: if the PLL range is set to zero, the PLL will operate at a low frequency (below 0.xxx GHz), if the PLL range is set to one, the PLL will operate at a high frequency (above 0.xxx GHz).

The **output slope** fields of the control register set the slew rate for the TTL outputs used for DRAM control, address and data signals, as detailed in a following section.

Mnemosyne uses a sufficiently high-frequency clock that internal SRAM timing can be controlled by synchronous logic, rather than asynchronous or self-timed logic. Internal SRAM timing may be controlled by loading values into configuration registers. The current specification reserves four bits for control of SRAM timing; one is currently used.

The **SRAM timing** bit is normally cleared, providing internal SRAM cycle time of 4 clock cycles. Setting the SRAM timing bit extends the cycle time to 5 clock cycles.

The **ECC seed** field of the control register provides a mechanism to cause ECC errors and thus test the ECC circuits. The field reserves 12 bits for this purpose, 8 bits are used in the current implementation. The field must be set to zero for

normal operation. The value of the field is xor'ed against the ECC value normally computed for write operation.

The **cidle 0** and **cidle 1** fields of the control register provide a mechanism to repeatedly sent simple patterns on the Hermes output channel for purposes of testing and skew adjustment. For normal operation, the **cidle 0** field must be set to zero (0), and the **cidle 1** field must be set to all ones (255).

MU 0023429

Status Register

The status register is a 64-bit register with both read and write access, though the only legal value which may be written is a zero, to clear the register. The result of writing a non-zero value is not specified.

The **reset/clear/selftest complete** bit of the status register complies with the Cerberus specification and is set upon the completion of a reset, clear or selftest operation as described above.

The **reset/clear/selftest status** bit of the status register complies with the Cerberus specification and is set upon the successful completion of a reset, clear or selftest operation as described above.

The **check byte error** bit of the status register is set when a received input packet has an incorrect check byte. The packet is otherwise ignored or forwarded to the Hermes output channel, and an error response packet is generated.

The **address error** bit of the status register is set when a received input request packet has an address which is not present on the device as currently configured. An error response packet is generated.

The **command error** bit of the status register is set when a packet is received on the Hermes input channel with an improper command, such as a read, write or error response packet.

The **uncorrectable ECC error** bit of the status register is set on the first occurrence of an uncorrectable ECC error in either the SRAM cache or the DRAM memory. The **ECC location flag** is set or cleared, indicating whether the error was in the cache memory (cleared, 0) or the DRAM memory (set, 1). The **ECC syndrome** field of the status register is loaded with the syndrome of the data for which the error was detected. The **ECC addr** register is loaded with the address of the data at which the error was detected. An error response packet is generated. Once one uncorrectable ECC error is detected, no further correctable or uncorrectable ECC errors are reported in the status register until this error is cleared by writing a zero value into the status register.

The **correctable ECC error** bit of the status register is set on the first occurrence of a correctable ECC error in either the SRAM cache or the DRAM memory, provided an uncorrectable ECC error has not already been reported. The **ECC location flag** is set or cleared, indicating whether the error was in the cache memory (cleared, 0) or the DRAM memory (set, 1). The **dirty flag** indicates, for an

error in the cache memory, the value of the dirty bit. The **ECC syndrome** field of the status register is loaded with the syndrome of the data for which the error was detected. The **ECC addr** register is loaded with the address of the data at which the error was detected. Once one uncorrectable ECC error is detected, no further correctable ECC errors are reported in the status register until this error is cleared by writing a zero value into the status register. The occurrence of this error will cause a response packet to be generated with a "stomped" check byte pattern, but is not explicitly reported with an error response packet.

The **other error** bit of the status register is set when errors not otherwise specified occur. There are no errors of this class reported by the current implementation.

The **PMOS drive strength** field of the status register is a read/only field that indicates the drive strength, or conductance gain, of PMOS devices on the Mnemosyne chip, expressed as a digital binary value. This field is used to calibrate the power and voltage level configuration, given variations in process characteristics of individual devices. The interpretation of the field is given by the table:

value	PMOS drive strength
0	Reserved
1	0.1*nominal
2	0.2*nominal
3	0.3*nominal
4	0.4*nominal
5	0.5*nominal
6	0.6*nominal
7	0.7*nominal
8	0.8*nominal
9	0.9*nominal
10	nominal
11	1.1*nominal
12	1.2*nominal
13	1.3*nominal
14	1.4*nominal
15	1.5*nominal

The **PLL in range** bit of the status register indicates that the Hermes input channel clock and the PLL oscillator are running at sufficiently similar rates such that the PLL can lock. This bit is used to verify or calibrate the settings of the **PLL range** field of the control register.

The **ECC location flag** bit of the status register, described above, indicates the location of an uncorrectable ECC error of a correctable ECC error. If the bit is set, the error was located in the DRAM memory, if the bit is clear, the error was located in the SRAM cache memory.

The **dirty flag** bit of the status register, described above, exhibits the dirty bit read from cache memory that results in an uncorrectable ECC error or correctable ECC error. The value is undefined if the currently reported ECC error was read from DRAM memory.

The ECC **syndrome** field of the status register, described above, exhibits the syndrome of an uncorrectable ECC error or correctable ECC error. A 12-bit field is reserved for this purpose; the current implementation uses eight bits of the field. The values in this field are implementation-dependent.

ECC **syndrome** values representing single-bit errors for MicroUnity's first implementation are detailed by the following table. Entries of * are not covered by the ECC code; syndrome values not shown in this table are uncorrectable errors involving two or more bits.

syndrome for **x**=	7	6	5	4	3	2	1	0
syndrome_{x+0}	128	64	32	16	8	4	2	1
data_{x+0}	127	124	122	121	118	117	115	112
data_{x+8}	158	157	155	152	151	148	146	145
data_{x+16}	174	173	171	168	167	164	162	161
data_{x+24}	191	188	186	185	182	181	179	176
data_{x+32}	206	205	203	200	199	196	194	193
data_{x+40}	223	220	218	217	214	213	211	208
data_{x+48}	239	236	234	233	230	229	227	224
data_{x+56}	254	253	251	248	247	244	242	241
$\text{addr}_{\mathbf{x}+0}$	*	*	*	*	*	*	*	*
$\text{addr}_{\mathbf{x}+8}$	62	61	59	*	*	*	*	*
addr_{x+16}	94	93	91	88	87	84	82	81
$\text{addr}_{\mathbf{x}+24}$							98	97
dirty bit								100

The **raw 0** and **raw 1** fields of the status register contain the values obtained from two adjacent samples of the Hermes input channel. The **raw 0** field contains a value obtained when the input clock was zero (0), and the **raw 1** field contains the value obtained on the immediately following sample, when the input clock was (1). Mnemosyne must ensure that reading the status register produces two adjacent samples, regardless of the timing of the status register read operation on Cerberus. These fields are read for purposes of testing and control of skew in the Hermes channel.

MU 0023431

ECC Address Register

The ECC **addr** register indicates the address at which an uncorrectable ECC error or correctable ECC error has occurred. Bits 63..2P+E of the ECC **addr** register are reserved; they read as 0. If the ECC **location flag** bit of the status register is zero, the ECC **addr** register contains the cache address in bits C-1..0, and the uncorrected cache tag in bits 2P+E-1..C.

DRAM Address Mapping

Mnemosyne may interleave up to 2^I DRAM accesses in order to provide for continuous access of the DRAM memory system at the maximum bandwidth of the DRAM data pins. At any point in time, while some memory devices are engaged in row precharge, others may be driving or receiving data, and others may be receiving row or column addresses. In order to maximize the utility of this interleaving, the logical memory address bits which select the DRAM bank are the least-significant bits.

A logical memory address determines which bank of DRAM is accessed, the row and column of such an access, and which interleave set is accessed. The diagram below shows the ordering of such fields in a general DRAM configuration; the bit addresses and field sizes shown are for a four-byte logical memory address and a two-way interleaved configuration of 1M-word DRAM devices.

31 22	21	20 11	10 1	0
0	sel	row	col	int
7	1	10	10	1

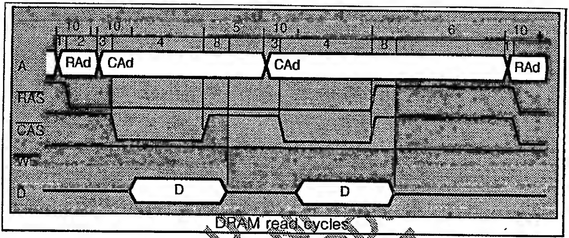
An access request which is decoded to contain the same values in the select, row, and int fields as a currently active request is queued until the completion of the active request, at which time the second request may be handled using a page mode access. This mechanism helps to maintain high bandwidth access even when the requests may not be perfectly interleaved, and provides for lower latency access in the event that the address stream is sufficiently local to take advantage of page mode access.

Mnemosyne devices may be cascaded for additional capacity, using the **ma** field in the packet formats. The memory controller must make the mapping between a contiguous address space and each of the separate address spaces made available within each Mnemosyne device. For maximum performance, the memory controller should also interleave such address spaces so that references to adjacent addresses are handled by different devices.

MU 0023432

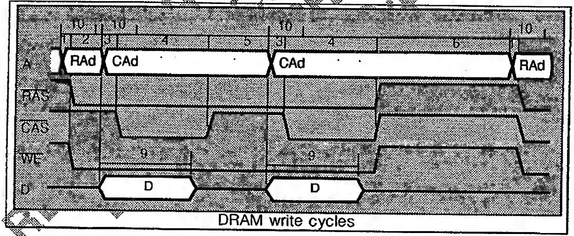
DRAM Timing Control

An internal state machine uses configurable settings to generate event timing, to accommodate DRAM performance variations. The timing of DRAM read cycles to a single DRAM bank is shown below:



DRAM read cycles

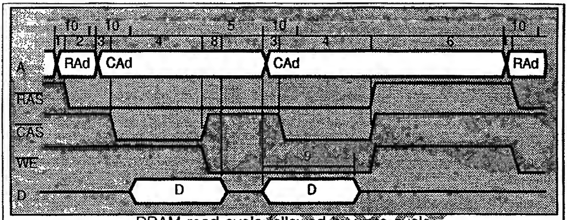
The timing of DRAM write cycles to a single DRAM bank is shown below:



DRAM write cycles

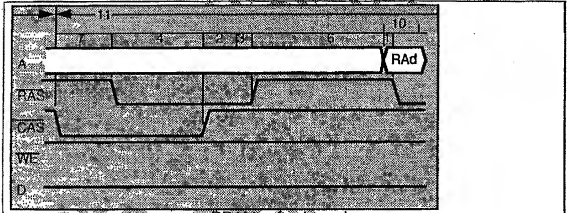
MU 0023433

The timing of a read cycle followed by a write cycle to a single DRAM bank is shown below:



DRAM read cycle followed by write cycle

The timing of a refresh cycle to a single DRAM bank is shown below:



DRAM refresh cycle

MU 0023434

Highly Confidential

The time intervals shown in the figures above control the following events:

interval	units	meaning
t1	4	Row address set up time relative to RAS.
t2	4	Row address hold time after RAS.
t3	4	Column address set up time relative to CAS.
t4	4	CAS pulse width. The data bus is sampled for a read cycle at the end of t4.
t5	4	Page mode cycle time is t3+t4+t5. Page mode CAS precharge is t3+t5.
t6	4	RAS precharge is t6+t1.
t7	4	CAS to RAS set up for refresh cycle. t7 >=t1 to ensure RAS precharge is met.
t8	4	Time data bus assumed to be occupied (by DRAM) after end of CAS low (end of t4) during read cycle. During t8, Mnemosyne will not drive CAS low for a read from another DRAM bank, or start a write cycle to another DRAM bank.
t9	4	Time data bus driven (by Mnemosyne) from column address drive (start of t3) during write cycle. During t9, Mnemosyne will not drive CAS low for a read from another DRAM bank, or start a write cycle to another DRAM bank.
t10	4	Interval between two address bus transitions. During t10, Mnemosyne will not change the address bus of another DRAM bank. This limits the noise generated by slewing the TTL address bus signals.
t11	1024	Interval between refresh cycles.

Additional DRAM operations may be requested before the corresponding DRAM bank is available, and are placed in a queue until they can be processed. Mnemosyne will queue DRAM writes with lower priority than DRAM reads, unless an attempt is made to read an address that is queued for a write operation. In such a case, DRAM writes are processed until the matching address is written. Mnemosyne may make an implementation-dependent pessimistic guess that such a conflict occurs, using a subset of the DRAM address to detect conflicts. The number of DRAM writes which are queued is implementation-dependent.

Mnemosyne uses one address bus for each interleave because dynamic power and noise is reduced by dividing the capacitance load of the DRAM address pins into four parts and only driving one-fourth of the load at a time. A timer (t10) prevents two address transitions from occurring too close together, to prevent power and noise on each address bus from having an additive effect. In addition, the loading of the already divided RAS, CAS, and WE signals is closer to the loading on the A signal when the address bus is also divided, reducing effects of capacitance loading on signal skew.

Power, Swing, Skew and Slew Calibration

Mnemosyne uses a set of configuration registers to control the power and voltage levels used for internal high-bandwidth logic and SRAM memory, to control skew in the output byte-channel, and to control slew rates in the TTL output circuits of the DRAM interface. The details of programming these registers are described below.

Eight-bit fields separately control the power and voltage levels used in a portion of the Mnemosyne circuitry. Each such field contains configuration data in the following format:

7	6 … 4	3 … 0
ov	**lvl**	**res**
1	3	4

power and swing controls

The range of valid values and the interpretation of the fields is given by the following table:

field	value	interpretation
ov	0..1	For global setting control, if set, turns off current sources in order to protect logic from damage during changes to voltage and resistor settings. This bit must be set prior to changing settings and cleared afterwards. For local setting control, If set, override these local settings by the global settings.
lvl	0..7	Set voltage swing level.
res	0..15	Set resistor load value.

Power and swing control field interpretation

Values and interpretations of the lvl field are given by the following table:

value	voltage swing level
0	
1	
2	
3	
4	
5	
6	
7	

Voltage swing level control field interpretation

MU 0023436

Values and interpretations of the **res** field are given by the following table:

value	resistor load value
0	Reserved
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

load value control field interpretation

When Mnemosyne is reset, a default value of 0 is loaded into each **ov** field, xxx in each **cur** field and xxx in each **res** field.

The **digital skew** fields set the number of delay stages inserted in the output path of the HoC and the Ho7..0 high-bandwidth output channel signals. Setting these fields, as well as the corresponding **analog skew** fields, permits a fine level of control over the relative skew between output channel signals. Nominal values for the output delay for various values of the digital skew and analog skew fields are given below:

digital skew	analog skew	delay (ps)
0	any	0
1	A[45]	135
	B	155
	C	175
	D	195
	E	215
2	A	220
	B	260
	C	300
	D	340
	E	380
3	A	330
	B	390
	C	450
	D	510
	E	570

When Mnemosyne is reset, a default value of 0 is loaded into the **digital skew** fields, setting a minimum output delay for the HoC and Ho7..0 signals.

[45]We need to get the right values for the analog skew setting to get these nominal values.

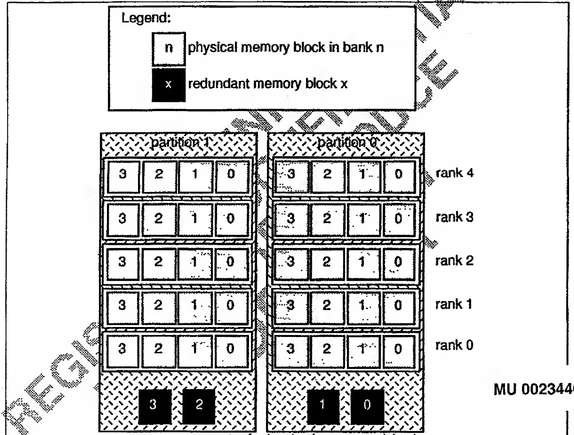
The **output slope** fields of the control register set the slew rate for the TTL outputs used for DRAM control, address and data signals, according to the following table:

setting	slew rate (V/ns) for control, address signals		slew rate (V/ns) for data signals	
	rising	falling	rising	falling
0				
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				

SRAM Redundancy Mapping

Mnemosyne uses a configurable set of redundant physical memory blocks to enhance the manufacturability of the cache memory. A systematic method for determining the proper configuration is described below.

To help clarify the following description, the figure below shows the logical arrangement of the physical memory blocks in the SRAM cache of MicroUnity's first Mnemosyne implementation. There are 40 physical memory blocks, each containing 2048 x 9 bits of data. The 40 blocks are divided into 4 banks of 10 blocks each. The 40 blocks are also divided into 2 partitions of 20 blocks each, and for each partition, there are two redundant memory blocks which can be configured to substitute for any of the 20 blocks in that partition. The 40 blocks are also divided up into 5 ranks, containing 8 blocks each, where each rank contains a distinct portion of a cache line. A cache line contains eight bytes of data, a 13-bit tag, a dirty bit, four unused bits, and an 8-bit ECC field.



arrangement of physical memory blocks for MicroUnity's first implementation

Each **redundant** x field, where x is in the range 0..**D*R**-1, controls the enabling and mapping address for a single redundant block. Starting at Cerberus address 32 and bits 63..56, each successive byte controls a redundant block, covering each redundant blocks in partition 0, and then in successive bytes, blocks for additional partitions. In other words, the **redundant** x field is located at Cerberus address $32+\lfloor\frac{x}{8}\rfloor$, bits 63-(x mod 8)..56-(x mod 8), and specifies the redundant mapping for block (x mod **R**), of redundant partition $\lfloor\frac{x}{R}\rfloor$. The format of each **redundant** x field

is detailed in the following figure, with bit field sizes shown for MicroUnity's first implementation:

7	6 5	4 2	1 0
en	**0**	**ra**	**ba**
1	2	3	2

redundant block controls

The range of valid values and the interpretation of the fields is given by the following table:

field	bits	value	interpretation
en	1	0..1	If set, use this redundant block to replace a physical memory block.
0	$7+\rfloor\log_2(\frac{\mathbf{D}}{\mathbf{N}})$	0	Pad control field to a byte.
ra	$-\rfloor\log_2(\frac{\mathbf{D}}{\mathbf{n}})$	$0..\frac{\mathbf{n}}{\mathbf{D}}-1$	Replace physical memory block at rank **ra** with the redundant block.
ba	$\log_2(\frac{\mathbf{N}}{\mathbf{n}})$	$0..\frac{\mathbf{N}}{\mathbf{n}}-1$	Replace physical memory block at bank **ba** with the redundant block.

Current and voltage control field interpretation

Redundancy is configured by first testing the SRAM cache with the **isolate/synch** bit if the control register set and all **redundant x** fields set to zero, and then again with each **redundant x** field set to 128+(x mod R). The result of the testing should indicate the location of all failures in the primary physical memory blocks and the redundant blocks. Then, each of the failed primary blocks is replaced with a working redundant block by setting the **redundant x** fields as required.

In order to map the address and bit identities of failures to physical block failures, the internal arrangement of bits and fields into blocks must be elaborated. First, a Mnemosyne memory address is divided into four parts according to the following figure, with bit field sizes shown for MicroUnity's first implementation:

31 26	25 13	12	1 0
0	**tag**	**ca**	**ba**
6	13	11	2

Mnemosyne cache address layout

MU 0023441

The interpretation of the fields is given by the following table:

field	bits	interpretation
0	8**A**-2**P**-**E**	Must be zero
tag	**t**	These bits are stored into the cache on a write operation and compared against bits read from the cache on a read operation.
ca	**C**-$\log_2 (\frac{\mathbf{N}}{\mathbf{n}})$	These bits are applied to the physical memory block to select a single SRAM cache word.
ba	$\log_2 (\frac{\mathbf{N}}{\mathbf{n}})$	These bits are used to select one of $\frac{\mathbf{N}}{\mathbf{n}}$ banks of physical memory blocks.

Mnemosyne cache address field interpretation

For each cache address and cache bank, a "line" of information, containing a cache tag, the cache data, and a dirty bit is stored. The internal arrangement of these fields is as shown in the following figure, with bit field sizes shown for MicroUnity's first implementation:

90 83	82 18	17	16 13	12 0
ECC	**data**	**d**	**u**	**tag**
8	64	1	4	13

Mnemosyne cache line layout
for MicroUnity's first implementation

The interpretation of the fields is given by the following table:

field	bits	interpretation
ECC	**e**	ECC bits used to correct single bit errors and detect multiple-bit errors.
data	8**W**	Data bits contain the visible cache data, as it appears in the packets.
d	1	Dirty bit: indicates that the cache line needs to be written to DRAM memory on a miss.
u	**S*****n**-**e**-8**W**-1-**t**	Unused bits pad cache line to even number of physical memory blocks.
tag	**t**	Tag bits identify a Mnemosyne logical address for this cache line.

Mnemosyne cache line field interpretation

MU 0023442

From the tables above, for each failure identified in the cache SRAM, a physical memory bank number, **ba**, can be identified from the Mnemosyne address, and a bit position, **bi**, can be identified from the Mnemosyne cache line layout. The bit position specifies a physical memory partition number, **pa**, according to the following formula:

$$\mathbf{pa} = \left\lfloor \frac{\mathbf{bi \bmod s*D}}{\mathbf{s}} \right\rfloor$$

90 83	82 18	17	16 13	12 0
ECC	**data**	**d**	**u**	**tag**
8	64	1	4	13

90 81	80 72	71 63	62 54	53 45	44 36	35 27	26 18	17 9	8 0
1	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**
9	9	9	9	9	9	9	9	9	9

Partition from bit position
for MicroUnity's first implementation

The bit position also identifies a physical memory block rank, **ra**, according to the following formula:

$$\mathbf{ra} = \left\lfloor \frac{\mathbf{bi}}{\mathbf{s*D}} \right\rfloor$$

90 83	82 18	17	16 13	12 0
ECC	**data**	**d**	**u**	**tag**
8	64	1	4	13

90 72	71 54	53 36	35 18	17 0
4	**3**	**2**	**1**	**0**
18	18	18	18	18

Rank from bit position
for MicroUnity's first implementation

So, to correct a failure in the cache SRAM, one of the working redundant blocks in the partition **pa** must be configured by setting a **redundant x** field, where **x** is in the range **pa*D+D-1..pa*D**, to the value:

7	6 5	4 2	1 0
1	**0**	**ra**	**ba**
1	2	3	2

redundant block controls

Multiple Memory Chips

Up to four Mnemosyne memory devices may be cascaded to form effectively larger memories. The cascade of memory devices will have the same bandwidth as a single memory chip, but more latency.

cascade of four memory devices

Packets are explicitly addressed to a particular Mnemosyne device; any packet received on a device's input channel which specifies another module address is automatically passed on via its output channel. This mechanism provides for the serial interconnection of Mnemosyne devices into strings, which function identically to a single Mnemosyne, except that a Mnemosyne string has larger memory capacity and longer response latency.

All devices in a cascade must have the same values for A and W parameters, in order that each part may properly interpret packet boundaries.

Response Packet Timing

In general, a received packet which is interpreted as a command causes a response packet to be generated. The latency between the end of the request packet and the beginning of the response packet is affected by the processing and forwarding of other packets, by the presence or absence of the requested word in the cache, by the setting of the SRAM and DRAM timing generators, by the presence of queued DRAM write and read requests, as well as other non-configurable and implementation-dependent device parameters.

With full knowledge of the cache state, configurable parameters and implementation-dependent characteristics, a memory controller may completely model the latency of responses. However, dependence on such characteristics is not recommended, except for testing and characterization purposes.

SRAM accesses, DRAM accesses, and forwarded packets typically have differing latency before a response or forwarded packet is generated at the Hermes output channel, so that certain combinations would imply that two output packets would need to overlap. In such a case, Mnemosyne will buffer the later output packet until such time as it can be transmitted. However, the number of requests that can be buffered is strictly limited to eight (the number of identification numbers) per Mnemosyne device. It is the responsibility of the issuer of command packets to ensure the number of outstanding packets never exceeds the limits of the buffer. Mnemosyne may use non-fair scheduling for forwarded packets to avoid buffer overflow conditions.

The use of DRAM page mode accesses and interleaving requires knowledge of the relationship between a pair of transactions. Therefore, additional DRAM requests per interleave level may be transmitted before the time at which the DRAM controller may perform the request. These additional requests are queued and the

corresponding response packet is generated at a time controlled by the DRAM timing generator. DRAM interleaves are serviced in an implementation-dependent fashion to ensure starvation-free scheduling.

Calliope Interface Architecture

Portions of this section has been temporarily removed to a separate document: "Calliope Interface Architecture," though it is still a mandatory area of the Terpsichore System Architecture.

MicroUnity's Calliope interface architecture is designed for ultra-high bandwidth systems. The architecture integrates fast communication channels with SRAM buffer memory and interfaces to standard analog channels.

The Calliope interfaces include byte-wide input and output channels intended to operate at rates of at least 1 GHz. These channels provide a packet communication link to synchronous SRAM memory on chip and a controller for interfaces to analog channels. Calliope provides analog interfaces for MicroUnity's Terpsichore system architecture. However, Calliope is useful in many interface applications.

Calliope's interface protocol embeds read and write operations to a single memory space into packets containing command, address, data, and acknowledgement. The packets include check codes that will detect single-bit transmission errors and multiple-bit errors with high probability. As many as eight operations in each device may be in progress at a time. As many as four Calliope devices may be cascaded to expand the buffer and analog interfaces.

Architecture Framework

The Calliope architecture builds upon MicroUnity's Hermes high-bandwidth channel architecture and upon MicroUnity's Cerberus serial bus architecture, and complies with the requirements of Hermes and Cerberus. Calliope uses parameters A and W as defined by Hermes.

The Calliope architecture defines a compatible framework for a family of implementations with a range of capabilities. The following implementation-defined parameters are used in the rest of the document in boldface. The value indicated is for MicroUnity's first Calliope implementation.

Parameter	Interpretation	Value	Range of legal values
C	$\log_2$ logical memory words in SRAM buffer	11	**C** ≥ 1
AI	number of AI audio inputs	1	**AI** ≤ 3
AO	number of AO audio outputs	1	**AO** ≤ 3
PO, PI	number of PO phone outputs and PI phone inputs	1	**PO** = **PI**, **PO** ≤ 3
VI	number of VI video inputs	1	**VI** ≤ 3
VO	number of VO video outputs	1	**VO** ≤ 3
IR, II	number of IR infrared outputs and II infrared inputs	1	**IR** = **II**, **IR** ≤ 3
SO, SI	number of SO smartcard outputs and SI smartcard inputs	1	**SO** = **SI**, **SO** ≤ 3
EQ, CI	number of EQ equalizers and CI cable inputs	2	**EQ** = **CI**, **EQ** ≤ 3
CO	number of CO cable outputs	2	**CO** ≤ 3
QPSK	number of QPSK cable inputs	1	**QPSK** ≤ 3

Interfaces and Block Diagram

Calliope uses two Hermes unidirectional, byte-wide, differential, packet-oriented data channels for its main, high-bandwidth interface between a memory control unit and Calliope's memory. This interface is designed to be cascadeable, with the output of a Calliope chip connected to the input of another, to expand the interface resources that can be reached via a single set of data channels. An external memory control unit is in complete control of the selection and timing of operations within Calliope and in complete control of the timing and content of information on the high-bandwidth interfaces.

A Cerberus bit-serial interface provides access to configuration, diagnostic and tester information, using TTL signal levels at a moderate data rate.

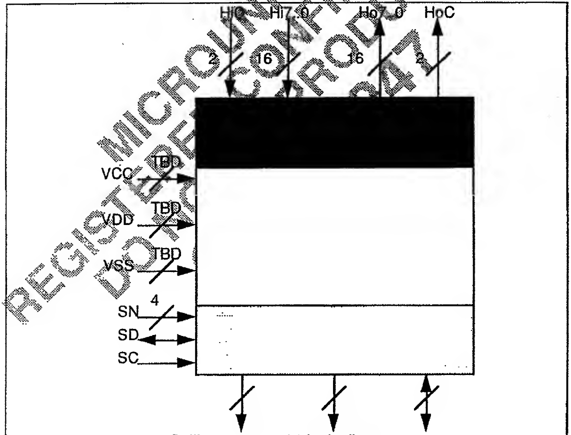
Nearly all Calliope circuits use a single power supply voltage, nominally at 3.3 Volts (5% tolerance). A second voltage of 5.0 Volts (5% tolerance) is used only for TTL interface circuits. Power dissipation is TBD. Initial packaging is TAB (Tape Automated Bonding).

Pin assignments are to be defined: there are 174 signal pins and 466 pins for 3.3V power, 5.0V power and substrate, for a total of 640 pins.

REDACTED

count	pin	meaning
18	HiC, $Hi_{7..0}$	hi-bandwidth input
18	HoC, $Ho_{7..0}$	hi-bandwidth output
6	SC, SD, $SN_{3..0}$	Cerberus interface
174		total signal pins
?	VDD	3.3 V above VSS
?	VCC[46]	5.0 V above VSS
?	VSS	most negative supply
640		total pins

The following is a diagram of the Calliope device interfaces. (Numerical values are shown for MicroUnity's first implementation.)



Calliope external block diagram

[46]Internal circuit documentation names this signal VDDO.

MU 0023448

Absolute Maximum Ratings	MIN	NOM	MAX	UNIT

Recommended operating conditions	MIN	NOM	MAX	UNIT	REF
V_T: Termination equivalent voltage	4.5	5.0	5.5	V	
Main supply voltage VDD	3.14	3.3	3.47	V	VSS
TTL supply voltage VCC	4.75	5.0	5.25	V	VSS
Operating free-air temperature	0		70	C	

Electrical characteristics	MIN	TYP	MAX	UNIT	REF
V_{OH}: H-state output voltage HoC, $Ho_{7..0}$				V	VDD
V_{OL}: L-state output voltage HoC, $Ho_{7..0}$				V	VDD
V_{IH}: H-state input voltage HiC, $Hi_{7..0}$				V	VDD
V_{IL}: L-state input voltage HiC, $Hi_{7..0}$				V	VDD
I_{OH}: H-state output current HoC, $Ho_{7..0}$				mA	
I_{OL}: L-state output current HoC, $Ho_{7..0}$				mA	
I_{IH}: H-state input current HiC, $Hi_{7..0}$				mA	
I_{IL}: L-state input current HiC, $Hi_{7..0}$				mA	
C_{IN}: Input capacitance HiC, $Hi_{7..0}$				pF	
C_{OUT}: Output capacitance HoC, $Ho_{7..0}$				pF	
~~V_{OH}: H-state output voltage $A_{11..0}$$_{3..0}$, $RAS_{3..0}$, $CAS_{3..0}$, $WE_{3..0}$, $DQ_{71..0}$~~	~~2.4~~		~~5.5~~	~~V~~	~~VSS~~
~~V_{OL}: L-state output voltage $A_{11..0}$$_{3..0}$, $RAS_{3..0}$, $CAS_{3..0}$, $WE_{3..0}$, $DQ_{71..0}$~~	~~0~~		~~0.4~~	V	~~VSS~~
V_{OL}: L-state output voltage SD	0		0.4	V	VSS
~~V_{IH}: H-state input voltage $DQ_{71..0}$~~	~~2.4~~		~~5.5~~	~~V~~	~~VSS~~
~~V_{IL}: L-state input voltage $DQ_{71..0}$~~	~~-0.5~~		~~0.8~~	~~V~~	~~VSS~~
V_{IH}: H-state input voltage SD	2.0		5.5	V	VSS
V_{IH}: H-state input voltage SC, $SN_{3..0}$	2.0		5.5	V	VSS
V_{IL}: L-state input voltage SC, SD, $SN_{3..0}$	-0.5		0.8	V	VSS
~~I_{OH}: H-state output current $A_{11..0}$$_{3..0}$, $RAS_{3..0}$, $CAS_{3..0}$, $WE_{3..0}$, $DQ_{71..0}$~~				~~µA~~	
~~I_{OL}: L-state output current $A_{11..0}$$_{3..0}$, $RAS_{3..0}$, $CAS_{3..0}$, $WE_{3..0}$, $DQ_{71..0}$~~			~~16~~	~~mA~~	
I_{OL}: L-state output current SD			16	mA	
I_{OZ}: Off-state output current SD	-10		10	µA	
~~I_{OZ}: Off-state output current $DQ_{71..0}$~~	~~-10~~		~~10~~	~~µA~~	
I_{IH}: H-state input current SC, $SN_{3..0}$	-10		10	µA	
I_{IL}: L-state input current SC, $SN_{3..0}$	-10		10	µA	
C_{IN}: Input capacitance SC, $SN_{3..0}$			4.0	pF	
C_{OUT}: Output or input-output capacitance, SD~~, $A_{11..0}$$_{3..0}$, $RAS_{3..0}$, $CAS_{3..0}$, $WE_{3..0}$, $DQ_{71..0}$~~			4.0	pF	

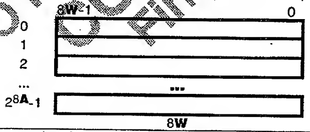
MU 0023450

Switching characteristics	MIN	TYP	MAX	UNIT
t_{BC}: HiC clock cycle time	1544			ps
t_{BCH}: HiC clock high time	600			ps
t_{BCL}: HiC clock low time	600			ps
t_{BT}: HiC clock transition time			100	ps
t_{BS}: set-up time, $Hi_{7..0}$ valid to HiC xition	200		100	ps
t_{BH}: hold time, HiC xition to $Hi_{7..0}$ invalid	-200		-100	ps
t_{OS}: skew between HoC and $Ho_{7..0}$	-50		50	ps
t_C: SC clock cycle time	50			ns
t_{CH}: SC clock high time	20			ns
t_{CL}: SC clock low time	20			ns
t_T: SC clock transition time			5	ns
t_S: set-up time, SD valid to SC rise				ns
t_H: hold time, SC rise to SD invalid				ns
t_{OD}: SC rise to SD valid	5			ns

Logical and Physical Memory Structure

Calliope defines two regions: a memory region, implemented by an on-device static RAM memory along with high-bandwidth control registers, and a configuration region, implemented by on-device read-only and read/write registers. These regions are accessed by separate interfaces: the Hermes channel used to access the memory region, and the Cerberus serial interface used to access the configuration region. These regions are kept logically separate.

The Calliope logical memory region is an array of 2^{8A} words of size W bytes. Each memory access, either a read or write, references all bytes of a single block. All addresses are block addresses, referencing the entire block.



Logical memory organization

Calliope's SRAM memory is a buffer for data which flows to or from interface devices.

Calliope's configuration region consists of read-only and read/write registers. The size of a logical block in the configuration memory space is eight bytes: one octlet.

Communications Channels

High-bandwidth

Calliope uses the Hermes high-bandwidth channel and protocols, implementing a slave device.

Calliope operates two Hermes high-bandwidth communications channels, one input channel and one output channel.

Calliope uses the Hermes packet structure. There is no structure corresponding to the Hermes-designated cache, so the no-allocate attribute of read and write operations has no effect..

Configuration-region registers provide a low-level mechanism to detect skew in the byte-wide input channel, and to adjust skew in the byte-wide output channel. This mechanism may be employed by software to adaptively adjust for skew in the channels, or set to fixed patterns to account for fixed signal skew as may arise in device-to-device wiring.

Serial

A Cerberus serial bus interface is used to configure the Calliope device, set diagnostic modes, and read diagnostic information, and to enable the use of the part within a high-speed tester.

The serial port uses the Cerberus serial bus interface.

Error Handling

Calliope performs error handling compliant with Hermes architecture.

For the current implementation, the following errors are designed to be detected and known not detected by design:

errors detected	errors not detected
invalid check byte	invalid identification number
invalid command	internal buffer overflow
invalid address	invalid check byte on idle packet
	uncorrectable error in SRAM buffer

Upon receipt of the error response packet, the packet originator must read the status register of the reporting device to determine the precise nature of the error. Calliope devices reporting an invalid packet will suppress the receipt of additional packets until the error is cleared, by clearing the status register. However, such devices may continue to process packets which have already been received, and

generate responses. Upon taking appropriate corrective actions and clearing the error, the packet originator should then re-send any unacknowledged commands.

Because of the large difference in clock rate between the high-bandwidth Hermes channel and the Cerberus serial bus interface, it is generally safe to assume that, after detecting an error response packet, an attempt to read the status register via Cerberus will result in reading stable, quiescent error conditions and that the queue of outstanding requests will have drained. After clearing the status register via Cerberus, the packet originator may immediately resume sending requests to the Calliope device.

Cerberus Registers

Calliope's configuration registers comply with the Cerberus and Hermes specifications. Cerberus registers are internal read/only and read/write registers which provide an implementation-independent mechanism to query and control the configuration of devices in a Terpsichore system. By the use of these registers, a user of a Terpsichore system may tailor the use of the facilities in a general-purpose implementation for maximum performance and utility. Conversely, a supplier of a Terpsichore system component may modify facilities in the device without compromising compatibility with earlier implementations. These registers are accessed via the Cerberus serial bus.

As a device component of a Terpsichore system, each Calliope interface contains a set of Cerberus-accessable configuration registers. Additional sets of configuration registers are present for each device in a Terpsichore system, including Euterpe processor devices, and Mnemosyne memory devices.

Read/only registers supply information about the Terpsichore system implementation in a standard, implementation-independent fashion. Terpsichore software may take advantage of this information, either to verify that a compatible implementation of Calliope is installed, or to tailor the use of the part to conform to the characteristics of the implementation.

The read/only registers occupy addresses 0..5. An attempt to write these registers may cause a normal or an error response.

Read/write registers select operating modes and select power and voltage levels for gates and signals. The read/write registers occupy addresses 6..7, 10..14 and 25..32.

Reserved registers in the range 8..9, 15..24 and 33..63 must appear to be read/only registers with a zero value. An attempt to write these registers may cause a normal or an error response.

Reserved registers in the range 64..2^{16}-1 may be implemented either as read/only registers with a zero value, or as addresses which cause an error response if reads or writes are attempted.

The format of the registers is described in the table below. The **octlet** is the Cerberus address of the register; **bits** indicate the posifion of the field in a register. The **value** indicated is the hard-wired value in the register for a read/only register, and is the value to which the register is initialized upon a reset for a read/write register. If a reset does not initialize the field to a value, or if initialization is not required by this specification, a * is placed in or appended to the value field. The **range** is the set of legal values to which a read/write register may be set. The **interpretation** is a brief description of the meaning or utility of the register field; a more comprehensive description follows this table.

octlet	bits	field name	value	range	interpretation
0	63..16	**architecture code**	0x00 40 a3 92 b4 49		Identifies interface device as compliant with MicroUnity Calliope architecture.
	15..0	**architecture revision**	0x01 00		Device complies with architecture version 1.0.

octlet	bits	field name	value	range	interpretation
1	63..16	**implementor code**	0x00 40 a3 49 db 3c		Identifies Calliope interface device as implemented by MicroUnity.
	15..0	**implementor revision**	0x01 00		Implementation version 1.0.

octlet	bits	field name	value	range	interpretation
2	63..16	**manufacturer code**	0x00 40 a3 a4 6d ff		Identifies initial manufacturer of Calliope interface device implemented by MicroUnity as MicroUnity.
	15..0	**manufacturer revision**	0x01 00		Manufacturing version 1.0.

octlet	bits	field name	value	range	interpretation
3	63..16	**serial number**	0		This device has no serial number capability.
	15..0	**dynamic address**	0		This device has no dynamic addressing capability.

octlet	bits	field name	value	range	interpretation
4	63..60	**A**	4	0..15	size of a Hermes address
	59..56	**log_2W**	3	0..15	size of a Hermes word
	55..48	**C**	11	0..255	log_2 of buffer capacity in words
	47..0	**0**	0	0	Reserved for definition in later revision of Calliope architecture

octlet	bits	field name	value	range	interpretation
5	63.20	**0**	0	0	Reserved for definition in later revision of Calliope architecture
	19..18	**AI**	1	0..3	number of AI audio inputs
	17..16	**AO**	1	0..3	number of AO audio outputs
	15..14	**PO, PI**	1	0..3	number of PO phone outputs and PI phone inputs
	13..12	**VI**	1	0..3	number of VI video inputs
	11..10	**VO**	1	0..3	number of VO video outputs
	9..8	**IR, II**	1	0..3	number of IR infrared outputs and II infrared inputs
	7..6	**SO, SI**	1	0..3	number of SO smartcard outputs and SI smartcard inputs
	5..4	**EQ, CI**	2	0..3	number of EQ equalizers and CI cable inputs
	3..2	**CO**	2	0..3	number of CO cable outputs
	1..0	**QPSK**	1	0..3	number of QPSK cable inputs

octlet	bits	field name	value	range	interpretation
6	63	**reset**	1	0..1	set to invoke device's circuit reset
	62	**clear**	1	0..1	set to invoke device's logic clear
	61	**selftest**	0	0..1	set to invoke device's selftest: bits 60..48 may indicate depth of selftest
	60	**defer writes**	0*	0..1	set to cause writes to octlets 25..43 to be deferred until the next logic-clear or non-deferred write.
	59..50	**0**	0	0	Reserved
	49..48	**module id**	0	0..3	Module identifier.
	47..33	**0**	0	0	Reserved
	32	**Hermes channel disable**	1	0..1	Set to cause Hermes input channel to be ignored and idles to be generated on output channel.
	31..16	**0**	0	0	Reserved
	15..8	**cidle 0**	0*	0..255	Value transmitted on idle Hermes output channel when output clock zero (0).
	7..0	**cidle 1**	255*	0..255	Value transmitted on idle Hermes output channel when output clock one (1).

MU 0023456

octlet	bits	field name	vfalue	range	interpretation
7	63	**reset/clear/ selftest complete**	1	0..1	This bit is set when a reset, clear or selftest operation has been completed.
	62	**reset/clear/ selftest status**	1	0..1	This bit is set when a reset, clear or selftest operation has been completed successfully.
	61	**meltdown detected**	0	0..1	This bit is set when the meltdown detector has caused a reset.
	60	**low voltage or temperature**	0	0..1	This bit is set when the voltage or temperature is too low for proper operation of logic circuits.
	59..57	**0**	0	0	Reserved for indicating additional causes of reset.
	56	**Cerberus transaction error**	0	0..1	This bit is set when a Cerberus transaction error has caused a machine check.
	55	**Hermes check byte error**	0	0..1	This bit is set when a Hermes channel check byte error has caused a machine check.
	54	**Hermes command error**	0	0..1	This bit is set when a Hermes channel command error has caused a machine check.
	53	**Hermes address error**	0	0..1	This bit is set when a Hermes address error has caused a machine check.
	52..16	**0**	0*	0	Reserved
	15..8	**raw 0**	*	0..255	Value sampled on specified Hermes channel when input clock is zero (0).
	7..0	**raw 1**	*	0..255	Value sampled on specified Hermes channel immediately following sample value in **raw 0** register.

octlet	bits	field name	value	range	interpretation
8..9	63..0	**0**	0	0	Reserved

MU 0023457

Highly Confidential

octlet	bits	field name	value	range	interpretation
10	63..56	**0**	0	0	Reserved
	55..48	**PLL anob**	224	0..230	PLL analog-knob settings
	47..40	**0**	0	0	Reserved
	39..32	**CI2 test**	0	0..7	CI2 test control
	31..24	**CI1 test**	0	0..7	CI1 test control
	23..16	**CI2adc anob**	224	0..230	CI2 ADC analog-knob settings
	15..12	**CI2Q filter**	3	0..7	CI2 Q filter adjust
	11..8	**CI2I filter**	3	0..7	CI2 I filter adjust
	7..4	**0**	0	0	Reserved
	3	**CI2 VCO**	0	0..1	CI2 external VCO switch
	2	**CI2 LNA**	0	0..1	CI2 input LNA enable
	1	**CI2Q ADC preamplifier**	0	0..1	CI2 Q ADC preamplifier disable
	0	**CI2I ADC preamplifier**	0	0..1	CI2 I ADC preamplifier disable

octlet	bits	field name	value	range	interpretation
11	63..56	**CI1syn anob**	224	0..230	CI1 synthesizer analog-knob settings
	55	**CO2 invert**	0	0..1	CO2 inversion control
	54	**CO1 invert**	0	0..1	CO1 inversion control
	53	**CI2a invert**	0	0..1	CI2a inversion control
	52	**CI2b invert**	0	0..1	CI2b inversion control
	51	**CI1a invert**	0	0..1	CI1a inversion control
	50	**CI1b invert**	0	0..1	CI1b inversion control
	49..48	**0**	0	0	Reserved
	47..40	**CI1adc anob**	224	0..230	CI1 ADC analog-knob settings
	39..36	**CI1Q filter**	3	0..7	CI1 Q filter adjust
	35..32	**CI1I filter**	3	0..7	CI1 I filter adjust
	31..28	**0**	0	0	Reserved
	27	**CI1 VCO**	0	0..1	CI1 external VCO switch
	26	**CI1 LNA**	0	0..1	CI1 input LNA enable
	25	**CI1Q ADC preamplifier**	0	0..1	CI1 Q ADC preamplifier disable
	24	**CI1I ADC preamplifier**	0	0..1	CI1 I ADC preamplifier disable
	23..16	**CI2syn anob**	224	0..230	CI2 synthesizer analog-knob settings
	15..8	**refclk anob**	224	0..230	reference clock divider analog-knob settings
	7..0	**CLIO anob**	224	0..230	CLIO analog-knob settings

octlet	bits	field name	value	range	interpretation
12	63	**capacitor calibration**	0	0..1	Set to enable capacitor calibration.
	62..34	**0**	0	0	Reserved
	33	**VI invert**	0	0..1	VI inversion control
	32	**VO invert**	0	0..1	VO inversion control
	31..24	**VI anob**	224	0..230	VI analog-knob settings
	23..16	**VO anob**	224	0..230	VO analog-knob settings
	15..8	**CO1 anob**	224	0..230	CO1 analog-knob settings
	7..0	**CO2 anob**	224	0..230	CO2 analog-knob settings

octlet	bits	field name	value	range	interpretation
13	63	**0**	0	0	Reserved
	62..56	**CO2 configuration**	0	0..127	CO2 configuration control
	55	**AI invert**	0	0..1	AI inversion control
	54	**PI invert**	0	0..1	PI inversion control
	53	**PO invert**	0	0..1	PO inversion control
	52	**AO invert**	0	0..1	AO inversion control
	51..50	**AIR bias**	2	0..3	AI right amplifier bias level
	51..48	**AIL bias**	2	0..3	AI left amplifier bias level
	47..40	**AIR anob**	224	0..230	AI right analog-knob settings
	39..32	**AIL anob**	224	0..230	AI left analog-knob settings
	31..26	**0**	0	0	Reserved
	25..24	**PI bias**	2	0..3	PI amplifier bias level
	23..16	**PI anob**	224	0..230	PI analog-knob settings:
	15..13	**0**	0	0	Reserved
	12	**mute**	1	0..1	AO and PO mute
	11..8	**PO filter**	7	0..15	PO antialias filter adjust
	7..4	**AOR filter**	7	0..15	AO right antialias filter adjust
	3..0	**AOL filter**	7	0..15	AO left antialias filter adjust

octlet	bits	field name	value	range	interpretation
14	63..56	**0**	0	0	Reserved
	55..48	**EQ2 test**	0	0..7	EQ2 test control
	47..40	**EQ1 test**	0	0..7	EQ1 test control
	39	**0**	0	0	Reserved
	38..32	**CO1 configuration**	0	0..127	CO1 configuration control
	31..16		*0*		left priority?
	15..0		*0*		right priority?

MU 0023459

Highly Confidential

octlet	bits	field name	value range		interpretation
15..24	63..0	**0**	0	0	Reserved for expansion of Cerberus registers upward or knobcity registers downward.

octlet	bits	field name	value range		interpretation
25	63..56		224	0..127	geographical digital knob settings
	55..48		224	0..127	geographical digital knob settings
	47..40		224	0..127	geographical digital knob settings
	39..32		224	0..127	geographical digital knob settings
	31..24		224	0..127	geographical digital knob settings
	23..16		224	0..127	geographical digital knob settings
	15..8		224	0..127	geographical digital knob settings
	7..0		224	0..127	geographical digital knob settings

octlet	bits	field name	value range		interpretation
26	63..56		224	0..127	geographical digital knob settings
	55..48		224	0..127	geographical digital knob settings
	47..40		224	0..127	geographical digital knob settings
	39..32		224	0..127	geographical digital knob settings
	31..24		224	0..127	geographical digital knob settings
	23..16		224	0..127	geographical digital knob settings
	15..8		224	0..127	geographical digital knob settings
	7..0		224	0..127	geographical digital knob settings

octlet	bits	field name	value range		interpretation
27	63..56		224	0..127	geographical digital knob settings
	55..48		224	0..127	geographical digital knob settings
	47..40		224	0..127	geographical digital knob settings
	39..32		224	0..127	geographical digital knob settings
	31..24		224	0..127	geographical digital knob settings
	23..16		224	0..127	geographical digital knob settings
	15..8		224	0..127	geographical digital knob settings
	7..0		224	0..127	geographical digital knob settings

octlet	bits	field name	value range		interpretation
28	63..56		224	0..127	geographical digital knob settings
	55..48		224	0..127	geographical digital knob settings
	47..40		224	0..127	geographical digital knob settings
	39..32		224	0..127	geographical digital knob settings
	31..24		224	0..127	geographical digital knob settings
	23..16		224	0..127	geographical digital knob settings
	15..8		224	0..127	geographical digital knob settings
	7..0		224	0..127	geographical digital knob settings

octlet	bits	field name	value range		interpretation
29	63..56		224	0..127	geographical digital knob settings
	55..48		224	0..127	geographical digital knob settings
	47..40		224	0..127	geographical digital knob settings
	39..32		224	0..127	geographical digital knob settings
	31..24		224	0..127	geographical digital knob settings
	23..16		224	0..127	geographical digital knob settings
	15..8		224	0..127	geographical digital knob settings
	7..0		224	0..127	geographical digital knob settings

octlet	bits	field name	value range		interpretation
29	63..56	**Hermes channel knob**	5	1..127	knob settings for Hermes channel circuits.
	55..48		224	0..127	geographical digital knob settings
	47..40		224	0..127	geographical digital knob settings
	39..32		224	0..127	geographical digital knob settings
	31..24		224	0..127	geographical digital knob settings
	23..16		224	0..127	geographical digital knob settings
	15..8		224	0..127	geographical digital knob settings
	7..0		224	0..127	geographical digital knob settings

octlet	bits	field name	value range		interpretation
30	63..62	**Hermes skew swing**	0	0..3	Voltage swing selection for Hermes channel skew circuits
	61..60	**0**	0	0	Reserved
	59..57	**resg**	5	0..7	Global resistor mask for all knobs.
	56..53	**0**	0	0	Reserved
	52..48	**termination fine-tuning**	20	0..31	Set based on value read from PMOS drive strength, used to fine-tune resistor values in Hermes termination.
	47..45	**0**	0	0	Reserved
	44..40	**process control**	20	0..31	Set based on value read from PMOS drive strength, used to fine-tune resistor values in knob settings.
	39..37	**0**	0	0	Reserved
	36..32	**PMOS drive strength**	*	0..31	This read/only field indicates the drive strength of PMOS devices expressed as a digital binary value.
	31..28	**swing 3**	15	0..15	Voltage swing knob setting 3
	27..24	**swing 2**	15	0..15	Voltage swing knob setting 2
	23..20	**swing 1**	15	0..15	Voltage swing knob setting 1
	19..16	**swing 0**	15	0..15	Voltage swing knob setting 0
	15..2	**reference 3**	15	0..15	Voltage reference knob setting 3
	11..8	**reference 2**	15	0..15	Voltage reference knob setting 2
	7..4	**reference 1**	15	0..15	Voltage reference knob setting 1
	3..0	**reference 0**	15	0..15	Voltage reference knob setting 0

octlet	bits	field name	value	range	interpretation
31	63..58	**0**	0	0	Reserved
	57	**PLL prescaler bypass**	0	0..1	Set to invoke PLL0 and PLL1 prescaler bypass, otherwise divide input clock by 20.
	56	**conversion prescaler bypass**	0	0..1	Set to invoke temperature conversion prescaler bypass, otherwise divide input clock by 20.
	55..51	**PLL2 divide ratio**	20	1..31	PLL2 divider ratio
	50	**PLL2 feedback bypass**	1*	0..1	Set to invoke PLL2 feedback bypass.
	49	**PLL2 range**	0*	0..1	Set for operation at high frequency (above 0.xxx GHz); cleared for operation at low frequency (below 0.yyy GHz).
	48	**PLL2 oscillator select**	0	0..1	Set to select multivibrator oscillator; cleared to select ring oscillator.
	47..43	**PLL1 divide ratio**	12	6..13	PLL1 divider ratio
	42	**PLL1 feedback bypass**	1*	0..1	Set to invoke PLL1 feedback bypass.
	41	**PLL1 range**	0*	0..1	Set for operation at high frequency (above 0.xxx GHz); cleared for operation at low frequency (below 0.yyy GHz).
	40	**PLL1 oscillator select**	0	0..1	Set to select multivibrator oscillator; cleared to select ring oscillator.
	39..35	**PLL0 divide ratio**	12	6..13	PLL0 divider ratio
	34	**PLL0 feedback bypass**	1	0..1	Set to invoke PLL0 feedback bypass.
	33	**PLL0 range**	0	0..1	Set for operation at high frequency (above 0.xxx GHz); cleared for operation at low frequency (below 0.yyy GHz).
	32	**PLL0 oscillator select**	0	0..1	Set to select multivibrator oscillator; cleared to select ring oscillator.
	31..24	**analog measurement**	0	0..255	Set to measure analog levels at various test points within device.

octlet	bits	field name	value	range	interpretation
	23..22	**meltdown threshold**	0	0..3	Set to perform margin testing of the meltdown detector.
	21	**conversion start**	0*	0..1	Setting this bit causes the conversion to begin. The bit remains set until conversion is complete
	20	**0**	0	0	Reserved. (selection extension)
	19..16	**conversion selection**	0*	0..9	Field selects which of ten measurements are taken
	15..10	**0**	0	0	Reserved. (counter extension)
	9..0	**conversion counter**	0*	0..1023	This field is set to the two's complement of the downslope count. The counter counts upward to zero, and then continues counting on the upslope until conversion completes.

octlet	bits	field name	value	range	interpretation
32	63	**0**	0	0	Reserved
	62	**quadrature bypass**	0*	0..1	Setting this bit causes the quadrature circuit to be bypassed; the input clock signal is used directly. with maximum delay.
	61	**quadrature range**	0*	0..1	Set to 0 if the Hermes channel is operating at a low frequency; 1 if the Hermes channel is operating at a high frequency.
	60	**output termination**	1	0..1	Set to enable output terminators. Cleared to disable output terminators.
	59..57	**termination resistance**	1	0..7	Set termination resistance level.
	56..54	**output current**	1	0..7	Set output current level.
	53..48	**skew bit 7**	1	0..63	Set delay in Ho7 skew circuit.
	47..42	**skew bit 6**	1	0..63	Set delay in Ho6 skew circuit.
	41..36	**skew bit 5**	1	0..63	Set delay in Ho5 skew circuit.
	35..30	**skew bit 4**	1	0..63	Set delay in Ho4 skew circuit.
	29..24	**skew bit 3**	1	0..63	Set delay in Ho3 skew circuit.
	23..18	**skew bit 2**	1	0..63	Set delay in Ho2 skew circuit.
	17..12	**skew bit 1**	1	0..63	Set delay in Ho1 skew circuit.
	11..6	**skew bit 0**	1	0..63	Set delay in Ho0 skew circuit.
	5..0	**skew clk**	1	0..63	Set delay in HoC skew circuit.

octlet	bits	field name	value	range	interpretation
33..63	63..0	**0**	0	0	Reserved for use with additional Hermes channel interfaces

octlet	bits	field name	value range		interpretation
64.. 65536	63..0	**0**	0	0	Reserved for use with later revisions of the architecture.
configuration memory space					

Identification Registers

The identification registers in octlets 0..3 comply with the requirements of the Cerberus architecture.

MicroUnity's company identifier is: 0000 0000 0000 0010 1100 0101.

MicroUnity's architecture code for Calliope is specified by the following table:

Internal code name	Code number
Calliope	0x00 40 a3 92 b4 49

Calliope architecture revisions are specified by the following table:

Internal code name	Code number
1.0	0x01 00

MicroUnity's Calliope implementor codes are specified by the following table:

Internal code name	Code number
MicroUnity	0x00 40 a3 49 db 3c

MicroUnity's Calliope, as implemented by MicroUnity, uses implementation codes as specified by the following table:

Internal code name	Revision number
1.0	0x01 00

MicroUnity's Calliope, as implemented by MicroUnity, uses manufacturer codes as specified by the following table:

Internal code name	Code number
MicroUnity	0x00 40 a3 a4 6d ff

MicroUnity's Calliope, as implemented by MicroUnity, and manufactured by MicroUnity, uses manufacturer revisions as specified by the following table:

Internal code name	Code number
1.0	0x01 00

Architecture Description Registers

The architecture description registers in octlets 4 and 5 comply with the Cerberus specification and contain a machine-readable version of the architecture parameters: **A, W, C, AI, AO, PO, PI, VI, VO, IR, II, SO, SI, EQ, CI, CO,** and **QPSK** described in this document.

The architecture parameters describe characteristics of the Hermes interface, capacity of the Calliope buffer memory, and the number of audio, phone, video, infrared, smartcard, and cable input and output channels, and the number of QPSK cable input channels.

Control Register

The control register in octlet 6 is a 64-bit register with both read and write access. It is altered only by Cerberus accesses; Calliope does not alter the values written to this register.

The **reset** bit of the control register complies with the Cerberus specification and provides the ability to reset an individual Calliope device in a system. Writing a one (1) to this bit is equivalent to a power-on reset or a broadcast Cerberus reset (low level on SD for 33 cycles) and resets configuration registers to their power-on values, which is an operating state that consumes nominal current (as determined by external pins), and also causes all internal high-bandwidth logic to be reset. The duration of the reset is sufficient for the operating state changes to have taken effect. At the completion of the reset operation, the **reset/clear/selftest complete** bit of the status register is set, the **reset/clear/selftest status** bit of the status register is set, and the **reset** bit of the control register is set.

The **clear** bit of the control register complies with the Cerberus specification and provides the ability to clear the logic of an individual Calliope device in a system. Writing a one (1) to this bit causes all internal high-bandwidth logic to be reset, as is required after reconfiguring power and swing levels. The duration of the reset is sufficient for any operating state changes to have taken effect. At the completion of the reset operation, the **reset/clear/selftest complete** bit of the status register is set, the **reset/clear/selftest status** bit of the status register is set, and the **clear** bit of the control register is set.

The **selftest** bit of the control register complies with the Cerberus specification and provides the ability to invoke a selftest on an individual Terpsichore device in a system. However, Calliope does not define a selftest mechanism at this time, so setting this bit will immediately set the **reset/clear/selftest complete** bit and the **reset/clear/selftest status** bit of the status register.

The **defer writes** bit of the control register provides a mechanism to adjust several octlets of Cerberus registers at one time with a single transition, such as when setting individual power levels within Calliope. Writing a one (1) to this bit causes writes to octlets 10 through 32 to have no effect (to be deferred) until the next logic-clear or a non-deferred write. When writes have been deferred, the values written are lost if a read of these octlets precedes the subsequent logic-clear or

non-deferred write. A normal or non-deferred write occurs when writing to octlets 10 through 32 while the **defer writes** bit is cleared (0).

The **module id**-field of the control register controls the value of the module identifier field of the Hermes input channel which selects this Calliope device.

The **Hermes channel disable** bit of the control register provides the means to begin operations on the Hermes channels after a reset, clear, or error. Writing a one (1) to this bit causes the Hermes input channel to be ignored and forces idles to be generated on the Hermes output channel. Writing a zero (0) to this bit causes the Hermes input channel phase adjustment to be reset, and after a suitable delay the Hermes channels are available for use.

The **cidle 0** and **cidle 1** fields of the control register provide a mechanism to repeatedly sent simple patterns on the Hermes output channel for purposes of testing and skew adjustment. For normal operation, the **cidle 0** field must be set to zero (0), and the **cidle 1** field must be set to all ones (255).

Status Register

The status register is a 64-bit register with both read and write access, though the only legal value which may be written is a zero, to clear the register. The result of writing a non-zero value is not specified.

The **reset/clear/selftest complete** bit of the status register complies with the Cerberus specification and is set upon the completion of a reset, clear or selftest operation as described above.

The **reset/clear/selftest status** bit of the status register complies with the Cerberus specification and is set upon the successful completion of a reset, clear or selftest operation as described above.

The **meltdown detected** bit of the status register is set when the meltdown detector has discovered an on-chip temperature above the threshold set by the **meltdown threshold** field of the Cerberus configuration register, which causes a reset to occur and the power level to be forced to minimum (1).

The **low voltage or temperature** bit of the status register is set when internal circuits have detected either insufficient voltage or temperature for proper operation of high speed logic circuits, which causes a logic clear until the condition is no longer detected (due to an increase in supply voltage or device temperature).

The **Cerberus transaction error** bit of the status register is set when a Cerberus transaction error (bus timeout, invalid transaction code, invalid address) has occurred. Note that Cerberus aborts, including locally detected parity errors, should cause bus retries, not a machine check.

The **Hermes check byte error** bit of the status register is set when a Hermes check byte error has occurred.

The **Hermes command error** bit of the status register is set when a Hermes command error has occurred.

The **Hermes address error** bit of the status register is set when a Hermes address error has occurred.

The **raw 0** and **raw 1** fields of the status register contain the values obtained from two adjacent samples of the specified Hermes input channel. The **raw 0** field contains a value obtained when the input clock was zero (0), and the **raw 1** field contains the value obtained on the immediately following sample, when the input clock was (1). Calliope ensures that reading the status register produces two adjacent samples, regardless of the timing of the status register read operation on Cerberus. These fields are read for purposes of testing and control of skew in the Hermes channel interfaces.

Power and Swing Calibration Registers

Calliope uses a set of calibration registers to control the power and voltage levels used for internal high-bandwidth logic and memory. The details of programming these registers are described below.

Eight-bit fields separately control the power and voltage levels used in a portion of the Calliope circuitry. Each such field used to control digital circuitry (labeled "knob") contains configuration data in the following format:

7 … 5	4 … 3	2 … 1	0
res	**lvl**	**ref**	**0**
3	2	2	1

power and swing controls

Each such field used to control analog circuitry (labeled "anob") contains configuration data in the following format:

7 … 5	4 … 3	2 … 1	0
res	**0**	**lvl**	**0**
3	2	2	1

power and swing controls

The range of valid values and the interpretation of the fields is given by the following table:

field	value	interpretation
0	0	Reserved
ref	0..3	Set reference voltage level
lvl	0..3	Set voltage swing level.
res	0..7	Set resistor load value.

Power and swing control field interpretation

MU 0023468

The reference voltage level, voltage swing level and resistor load value are model figures for a full-swing, lowest-power logic gate output. The actual voltage levels and resistor load values used in various circuits is geometrically related to the values in the tables below. Designed typical, full-speed settings for the **ref**, **lvl** and **res** fields are **ref**=250 millivolts, **lvl**=500 millivolts, and **res**=2.5 kilohms.

The **ref** field, together with the **reference n** fields of the configuration register, control the reference voltage level used for logic circuits in the specified knob domain. The value of the **ref** field is interpreted by the following table:

ref	reference voltage level
0	reference 0
1	reference 1
2	reference 2
3	reference 3

The **lvl** field, together with the **swing n** fields of the configuration register, control the voltage swing level used for logic circuits in the specified knob domain. The value of the **lvl** field is interpreted by the following table:

lvl	voltage swing level
0	swing 0
1	swing 1
2	swing 2
3	swing 3

Values and interpretations of the **swing n** and **reference n** fields are given by the following table, with units in millivolts:

value	reference	swing
0	138	275
1	150	300
2	163	325
3	175	350
4	188	375
5	200	400
6	213	425
7	225	450
8	238	475
9	250	500
10	263	525
11	275	550
12	288	575
13	300	600
14	325	650
15	350	700

REDACTED

The res field, together with the **resg** field of the configuration register and the **meltdown detected** bit of the status register, control the PMOS load resistance value used for logic circuits in the specified knob domain, referred here as the **resl** value. For each res field, the resl value is computed as:

resl = **res** & (**meltdown detected** ? 1 : **resg**)

The **resl** value, together with the **process control** field of the configuration register, control the PMOS load resistance value used for logic circuits in the specified knob domain. Values and interpretations of the **lvl** field are given by the following table, with units in kilohms. The table below gives resistance values with nominal process parameters.

	process control							
resl	0	4	8	12	16	20	24	28
0	undefined							
1		2.5	5.0	7.5	10	13	15.	18.
2		1.3	2.5	3.8	5.0	6.3	7.5	8.8
3		.83	1.7	2.5	3.3	4.2	5	5.8
4		.63	1.3	1.9	2.5	3.1	3.8	4.4
5		.50	1.0	1.5	2.0	2.5	3	3.5
6		.42	.83	1.3	1.7	2.1	2.5	2.9
7		.36	.71	1.1	1.4	1.8	2.1	2.5

Resistor control field interpretation

When the **process control** field of the configuration register is set equal to the PMOS drive strength field of the configuration register, nominal PMOS load resistance values are as given by the following table, with units in kilohms.

res	**PMOS load resistance**
0	undefined
1	13.
2	6.3
3	4.2
4	3.1
5	2.5
6	2.1
7	1.8

MU 0023470

When Mnemosyne is reset, a default value of 0 is loaded into each **0** field, 0 in each **ref** field, 0 in each **lvl** field and 7 in each **res** field, which is a byte value of 224. The **process control** field of the configuration register is set to 20, and the **reference n** and **swing n** fields are set to 15. These settings correspond to a chip with nominal processing parameters, nominal power and high voltage swing operation.

For nominal operating conditions, the **ref** field is set to 0, the **lvl** field is set to 0, and the res field is set to 5, which is byte value of 5. The **process control** field is set

equal to the **PMOS strength** field, and the **reference n** and **swing n** fields are set to 5.

interface Configuration Registers

Interface configuration registers are provided on the Calliope interface to control the [insert summary list of controls].

The **CI1 test** and **CI2 test** field of interface configuration register 10 control operating modes of the CI1 and CI2 cable input blocks.

Eight-bit fields separately control the operating modes of the cable input blocks. Each such field contains configuration data in the following format:

7 … 4	3	2	1	0
0	**rotate**	**round**	**test**	**DSP enable**
3	1	1	1	1

cable input test controls

The range of valid values and the interpretation of the fields is given by the following table:

field	value	interpretation
0	0	Reserved
rotate	0..1	Set to enable rotator
round	0..1	Set to enable multiplier rounding
test	0..1	Set to bypass ADC and connect cable input to cable output (digital loop back)
DSP enable	0..1	Set to enable DSP output (clear to enable testing of RAM)

Cable input test control field interpretation

The **CI1Q filter**, **CI1I filter**, **CI2Q filter** and **CI2I filter** fields of interface configuration register 10 and 11 control the cutoff frequency of the cable input antialias filters.

Four-bit fields separately control the cutoff frequency of each cable input antialias filter. Each such field contains configuration data in the following format:

3	2 … 0
0	**cutoff**
1	3

cable input antialias filter controls

MU 0023471

Highly Confidential

The range of valid values and the interpretation of the fields is given by the following table:

field	value	interpretation
0	0	Reserved
cutoff	0..7	Cutoff frequency selection for antialias filter

Cable input antialias filter control field interpretation

Values and interpretations of the **cutoff** fields are given by the following table, with units in megahertz, for nominal 3 dB frequency, at specified junction temperature:

cutoff	25 C	75 C	125 C
0	14.1	13.8	13.4
1	11.9	11.7	11.4
2	10.4	10.2	10.0
3	9.2	9.1	8.9
4	8.3	8.2	8.0
5	7.6	7.5	7.3
6	7.0	6.9	6.7
7	6.4	6.4	6.2

For normal operation a value of 3 is placed in the **cutoff** fields, selecting a 9 MHz cutoff frequency.

The **CI1 VCO** and **CI2 VCO** bits of interface configuration registers 10 and 11 control the selection of the VCO used as an input to the tuner of the cable input. Writing a zero (0) to the bit selects the internal VCO, while writing a one (1) selects an external VCO input. In normal operation a zero is placed in the **VCO** bit, selecting the internal VCO.

The **CI1 LNA** and **CI2 LNA** bits of interface configuration registers 10 and 11 enable the LNA (low noise amplifier) used as an input to the tuner of the cable input. Writing a zero (0) to the bit disables the LNA, while writing a one (1) enables the LNA. In normal operation a one is placed in the **LNA** bit, enabling the LNA.

The **CI1Q ADC preamp, CI1I ADC preamp, CI2Q ADC preamp** and **CI2I ADC preamp** bits of interface configuration registers 10 and 11 enable the ADC preamplfier output used as an input to the ADC of the cable input. Writing a zero (0) to the bit enables the ADC preamplifier output, while writing a one (1) disables it, allowing the tuner input to be driven from an external pin. In normal operation a zero is placed in the **ADC preamp** bits, enabling the preamplifiers.

The **CI1a, CI1b, CI2a, CI2b, CO1, CO2, VI, VO, AI, AO, PI, PO invert** bits of interface configuration registers 11, 12, and 13 provide for the selective inversion of the relative clock phase of the analog-to-digital section internal interfaces in the

respective interfaces. In normal operation, a zero is placed in the **invert** bits, matching the relative phases of the interface sections.

The **CO1 configuration** and **CO2 configuration** fields of interface configuration registers 13 and 14 provide for the configuration of external devices which assist in the implementation of the cable output. The configuration fields drive LVTTL outputs which can control external filters and other components. In nornal operation, a zero is placed in the **configuration** fields.

The **PI bias, AIR bias** and **AIL bias** fields of interface configuration register 13 control the bias current of the phone and audio input right and left operational amplifiers.

Four-bit fields separately control the bias current of each input operational amplifier. Each such field contains configuration data in the following format:

1 0
bias
2

audio input operational amplifier controls

The range of valid values and the interpretation of the fields is given by the following table:

field	value	interpretation
bias	0..3	bias current selection for input operational amplifier

audio input operational amplifier control field interpretation

Values and interpretations of the **bias** fields are given by the following table, with units in microamperes for nominal current at specified junction temperature:

bias	25 C	75 C	125 C
0		200	
1		133	
2		100	
3		80	

The **mute** bit of interface configuration register 13 provides for initial muting of the audio and phone outputs during initial system operation. Writing a zero (0) to the bit enables the audio and phone outputs, while writing a one (1) forces the AO and PO outputs to a constant value (zero with AC coupling).

The **PO filter, AOR filter** and **AOL filter** fields of interface configuration register 13 control the cutoff frequency of the phone and audio output right and left antialias filters.

Four-bit fields separately control the cutoff frequency of each output antialias filter. Each such field contains configuration data in the following format:

3 0
cutoff
4

cable input antialias filter controls

The range of valid values and the interpretation of the fields is given by the following table:

field	value	interpretation
cutoff	0..15	Cutoff frequency selection for antialias filter

audio output antialias filter control field interpretation

Values and interpretations of the cutoff fields are given by the following table, with units in kilohertz, for nominal 3 dB frequency at specified junction temperature:

cutoff	25 C	75 C	125 C
0		69.9	
1		65.9	
2		62.3	
3		58.8	
4		55.8	
5		53.2	
6		50.9	
7		48.6	
8		46.4	
9		44.6	
10		43.9	
11		41.4	
12		40.0	
13		38.5	
14		37.2	
15		35.9	

The EQ1 test and EQ2 test field of interface configuration register 14 control operating modes of the EQ1 and EQ2 cable input equalizers.

Eight-bit fields separately control the operating modes of the cable input equalizers. Each such field contains configuration data in the following format:

7 … 3	2	1	0
0	**round**	**0**	**DSP enable**
3	1	1	1

cable input equalizer test controls

The range of valid values and the interpretation of the fields is given by the following table:

field	value	interpretation
0	0	Reserved
round	0..1	Set to enable multiplier rounding
DSP enable	0..1	Set to enable DSP output (clear to enable testing of RAM).

Cable input equalizer test control field interpretation

Configuration Register

A Configuration register is provided on the Calliope interface to control the fine-tuning of the Hermes channel configuration, to control the global process parameter settings, to control the two phase-locked loop frequency generators, and to control the temperature sensors and read temperature values.

The Hermes **skew swing** field of the configuration register control the voltage swing used in the Hermes channel skew circuits. The field should always be set equal to the value of the **lvl** subfield of the **Hermes channel knob** field.

The **resg** field of the configuration register permits the global control of the load resistors in all of Calliope's high-speed logic circuits. The **resg** field is initially loaded from external pins to a numinal power level (5), and can be changed again to a value in the range 0..7 to lower or raise the power and speed of the high-speed logic citcuits in the Calliope device, or can be set to all ones (7) to enable control of individual sections of the Calliope device power levels. By altering the value on the external pins, Calliope can be configured for low-power (0 or 1) testing in a restricted packaging environment.

The **termination fine-tuning** field of the configuration register controls the analog bias settings for PMOS loads in Hermes termination circuits, in order to accomodate variations in circuit parameters due to the manufacturing process, and to provide intermediate termination resistance levels. Under normal operating conditions, the value read from the **PMOS drive strength** field should be written into the **termination fine-tuning** field. The interpretation of the field is given by the table:

value	termination fine-tuning
0	Reserved
1-19	increase PMOS conductance to 20/value*nominal.
20	use PMOS loads at nominal conductance.
21-31	decrease PMOS conductance to 20/value*nominal.

The **process control** field of the configuration register controls the analog bias settings for PMOS loads in internal logic circuits, in order to accomodate variations in circuit parameters due to the manufacturing process. Under normal operating conditions, the value read from the **PMOS drive strength** field should be written into the **process control** field. The interpretation of the field is given by the table:

value	process control
0	Reserved
1-19	increase PMOS conductance to 20/value*nominal.
20	use PMOS loads at nominal conductance.
21-31	decrease PMOS conductance to 20/value*nominal.

MU 0023476

Highly Confidential

The **PMOS drive strength** field of the configuration register is a read/only field that indicates the drive strength, or conductance gain, of PMOS devices on the Terpsichore chip, expressed as a digital binary value. This field is used to calibrate the power and voltage level configuration, given variations in process characteristics of individual devices. The interpretation of the field is given by the table:

value	PMOS drive strength
0	Reserved
1-19	value/20*nominal conductance
20	nominal conductance
21-31	value/20*nominal conductance

MU 0023477

There are two identical phase locked-loop (PLL) frequency generators, designated PLL0 and PLL1. These PLLs generate internal and external clock signals of configurable frequency, based upon an input clock reference of either 54 MHz or 1.08 GHz. PLL0 controls the internal operating frequency of the Terpsichore processor, while PLL1 controls the operating frequency of the Hermes channel interfaces. The configuration fields for PLL0 and PLL1 have identical meanings, described below.

The **PLL0 divide ratio** and **PLL1 divide ratio** fields select the divider ratio for each PLL, where legal values are in the range 6..21, with a nominal setting of 12 for PLL0, and 20 for PLL1. These divider ratios permit clock signals to be generated in the range from 324 MHz to 1.134 GHz, when the input clock reference is at 54 MHz, with prescaling bypassed, or at 1.08 GHz with prescaling used.

Setting the **PLL0 feedback bypass** bit or the **PLL1 feedback bypass** bit of the configuration register causes the generated clock bypass the PLL oscillator and to operate off the input clock directly. Setting these bits causes the frequency generated to be the optionally prescaled reference clock. These bits are cleared during normal operation, and set by a reset.

The **PLL0 range** field and the **PLL1 range** field of the configuration register are used to select an operating range for the internal PLLs. If the PLL range is set to zero, the PLL will operate at a low frequency (below 0.xxx GHz), if the PLL range is set to one, the PLL will operate at a high frequency (above 0.xxx GHz). At reset this bit is cleared, as the input clock frequency is unknown.

Setting the **PLL prescaler bypass** bit of the configuration register causes the phase-locked loops PLL0 and PLL1 to use the input clock directly as a reference clock. This bit is cleared during normal operation with a 1.08 GHz input clock, in which the input clock is divided by 20, and is set during normal operation with a 54 MHz input clock. At reset this bit is cleared, as the input clock frequency is unknown.

Setting the **conversion prescaler bypass** bit of the configuration register causes the temperature conversion unit to use the input clock directly as a reference clock.

REDACTED

Otherwise, clearing this bit causes the input clock to be divided by 20 before use as a reference clock. The reference clock frequency of the temperature conversion unit is nominally 54 MHz, and in normal operation, this bit should be set or cleared, depending on the input clock frequency. At reset this bit is cleared, as the input clock frequency is unknown.

The **meltdown margin** field controls the setting of the threshold at which meltdown is signalled. This field is used to test the meldown prevention logic. The interpretation of the field is given by the table below with a tolerance of ±6 degrees C, and 5 degrees C hysteresis:

value	meltdown threshold
0	150 degrees C
1	90 degrees C
2	50 degrees C
3	20 degrees C

The **conversion start** bit controls the initiation of the conversion of a temperature sensor or reference to a digital value. Setting this bit causes the conversion to begin, and the bit remains set until conversion is complete, at which time the bit is cleared.

The **conversion selection** field controls which sensor or reference value is converted to a digital value. The interpretation of the field is given by the table below:

value	conversion selected
0	local temperature sensor
1	local temperature reference
2..15	Reserved

The **conversion counter** field is set to the two's complement of the downslope count. The counter counts upward to zero, at which point the upslope ramp begins, and continues counting on the upslope until the conversion completes.

Hermes channel Configuration Registers

Configuration registers are provided on the Calliope interface to control the timing, current levels, and termination resistance for the Hermes channel high-bandwidth channel. A configuration register at octlet 31 is dedicated to the control of the Hermes channel, and additional information in the configuration register at octlet 31 controls aspects of the Hermes channel circuits in common. The Hermes channel configuration registers are Cerberus registers 32, where 32 corresponds to Hermes channel 0.

The **quadrature bypass** bit controls whether the HiC clock signal is delayed by approximately $\frac{1}{4}$ of a HiC clock cycle to latch the $Hi_{7..0}$ bits. In normal, full speed operation, this bit should be cleared to a zero value. If this bit is set, the

quadrature delay is defeated and the HiC clock signal is used directly to latch the $Hi_{7..0}$ bits.

The **quadrature range** bit is used to select an operating range to the quadrature delay circuit. If the quadrature range is set to zero, the circuit will operate at a low frequency (below 0.xxx GHz), if the quadrature range is set to one, the circuit will operate at a high frequency (above 0.xxx GHz).

The **output termination** bit is used to select whether the output circuits are resistively terminated. If the bit is set to a zero, the output has high impedence; if the bit is set to one, the output is terminated with a resistance equal to the input termination. At reset, this bit is set to one, terminating the output.

The **termination resistance** field is used to select the impedence at which the Hermes channel inputs, and optionally the Hermes channel outputs are terminated. The resistance level is controlled relative to the setting of the **termination fine tuning** field of the configuration register. The interpretation of the field is given by the table, with units in Ohms and nominal PMOS conductance and bias settings:

value	termination resistance
0	Reserved
1	250. Ohms
2	125. Ohms
3	83.3 Ohms
4	62.5 Ohms
5	50.0 Ohms
6	41.7 Ohms
7	35.7 Ohms

The **output current** field is used to select the current at which the Hermes channel outputs are operated. The interpretation of the field is given by the table, with units in mA:

value	output current
0	Reserved
1	2. mA
2	4. mA
3	6. mA
4	8. mA
5	10. mA
6	12. mA
7	14. mA

MU 0023479

The output voltage swing is the product of the composite termination resistance: (input termination resistance^{-1}+output termination resistance^{-1})$^{-1}$, and the output current. The output voltage swing should be set at or below 700 mV, and is

normally set to the lowest value which permits a sufficiently low bit error rate, which depends upon the noise level in the system environment.

The **skew** fields individually control the delay between the internal Hermes channel output clock and each of the HoC and Ho7..0 high bandwidth output channel signals. Each skew field contains two three-bit values, named digital skew and analog skew as shown below:

5 3	2 0
digital skew	**analog skew**
3	3

The **digital skew** fields set the number of delay stages inserted in the output path of the HoC and the Ho7..0 high-bandwidth output channel signals. The **analog skew** fields control the power level, and thereby control the switching delay, of a single delay stage. Setting these fields permits a fine level of control over the relative skew between output channel signals. Nominal values for the output delay for various values of the digital skew and analog skew fields are given below, assuming a nominal setting for the Hermes channel knob.

digital skew	delay (ps)	plus analog skew
0	0	no
1	320	yes
2	400	yes
3	470	yes
4	570	yes
5	670	yes
6	770	yes
7	870	yes

analog skew	delay (ps)
0	Reserved
1	???
2	???
3	+40
4	+20
5	0
6	-10
7	-20

When Calliope is reset, a default value of 0 is loaded into the **digital skew** and 1 is loaded into the **analog skew** fields, setting a minimum output delay for the HoC and Ho7..0 signals.

Hermes High-Bandwidth Channel

MicroUnity's Hermes high-bandwidth channel architecture is designed to provide ultra-high bandwidth communications between devices within MicroUnity's Terpsichore system architecture.

Hermes-compliant devices include one or more byte-wide input and output channels intended to operate at rates of at least 1 GHz. These channels provide a packet communication link to general devices, processors, memories, and input-output interfaces.

Hermes high-bandwidth channels employ nine signals, one clock signal and eight data signals, using differential low-voltage levels for direct communication from one device to another. The channels are designed to be arranged into a ring consisting of up to four target devices and one initiator. The channels may also be extended to permit multiple initiators in a single ring.

The Hermes interface protocol embeds read and write operations to a single memory space into packets containing command, address, data, and acknowledgement. The packets include check codes that will detect single-bit transmission errors and multiple-bit errors with high probability. As many as eight operations in each device may be in progress at a time. As many as four Hermes devices may be cascaded to expand system capacity and bandwidth.

Hermes relies upon MicroUnity's Cerberus serial bus to provide access to a low-level mechanism to detect skew in input channels, and to adjust skew in output channels. This mechanism may be employed by software to adaptively adjust for skew in the channels, or set to fixed patterns to account for fixed signal skew as may arise in device-to-device wiring.

Architecture Framework

The Hermes architecture defines a compatible framework for a family of implementations with a range of capabilities. The following implementation-defined parameters are used in the rest of the document in boldface. The value indicated are for MicroUnity's first implementations.

Parameter	Interpretation	Value	Range of legal values
A	$\log_{256}$ words in logical memory space or size in bytes of a logical memory address	4	$1 \leq \mathbf{A} \leq 8$
W	size in bytes of logical memory word	8	$1 \leq \mathbf{W} \leq 2^{15}$, $\log_2 \mathbf{W} \in Z$

Hermes devices have several optional capabilities, which are identified in the following table:

REDACTED

Capability	Meaning
Master	Capable of generating requests on output channel and receiving responses on input channel
Slave	Capable of receiving requests on input channel and generating responses on output channel
Forwarding	Capable of forwarding requests and responses from input channel to output channel
Cache	Capable of storing values previously read or written and returning these values on subsequent reads.

Electrical Signalling

Each Hermes channel consists of a one byte wide data path and a single-phase, constant-rate clock signal. Both the data and clock signals are differential-pair signals. The clock signal contains alternating zero and one values transmitted with the same timing as the data signals; thus, the clock signal frequency is one-half the channel byte data rate.

Each channel runs at a constant frequency and contains no auxiliary control, handshaking, or flow-control information. The channel transmitter is responsible for transmitting all nine differential-pair signals so as to be received with minimal skew; the receiver is responsible for decoding the signals in the presence of noise and skew as may arise due to differences in the signal environment of the clock and of each data bit.

A Hermes device may be capable of responding to Hermes request packets received on a Hermes input channel. Such a device is designated a slave device, and must operate the Hermes output channel at the same clock rate as the input channel. A slave device must generate no more than a specified amount of variation in the output clock phase, relative to the input clock, over changes in system temperature or operating voltage.

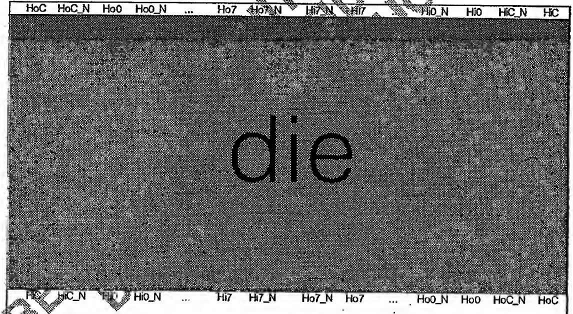
A Hermes device that is capable of generating Hermes request packets is designated a master device. A master device must be capable of generating the constant-frequency clock signal on the Hermes output channel and accepting signals on the Hermes input channel at the same clock frequency as is generated. In addition, a master device must accept an arbitrary input clock phase, and must accept a specified amount of variation in clock phase, as may arise due to changes in system temperature or operating voltage.

Each Hermes input or output channel requires 18 pads, and the associated Cerberus interface requires an additional 6 pads.

count	pad	meaning
18	HiC, HiC_N, $Hi_{7..0}$, $Hi_{7..0}$_N	Hermes input channel
18	HoC, HoC_N, $Ho_{7..0}$, $Ho_{7..0}$_N	Hermes output channel
6	SC, SD, $SN_{3..0}$	Cerberus interface
36c+6		total signal pads

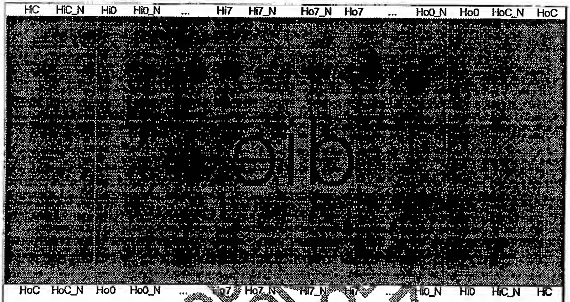
Each Hermes input channel is terminated at a nominal 50 ohm impedance to ground. Each Hermes output channel is optionally terminated at the same impedance as the devices input channel. An adjustable termination impedance, programmable via Cerberus is recommended.

In order to provide for planar connections among Hermes devices when connected into rings, all devices must locate Hermes input channels and Hermes output channels to pin assignments which preserve the following ordering, when viewed from the top of the device die:



Hermes device interfaces, face of die

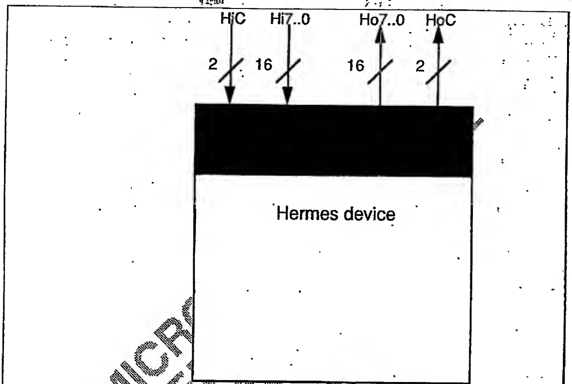
Hermes devices are generally designed to be placed on circuit boards face-down, so when viewed from the top of the circuit board, this becomes the ordering:



Hermes device interfaces, die face-down on circuit board

Other packaging systems may mount Hermes devices in a face-up orientation, if this is the case, all devices must be mounted in the same face-up orientation to avoid requiring vias and/or crossovers in the connections.

The following is a diagram of the Hermes and Cerberus device interfaces, for a device with a single pair of Hermes channel interfaces.



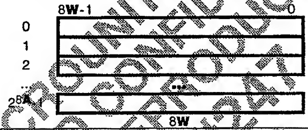
Hermes device interfaces

Electrical characteristics	MIN	TYP	MAX	UNIT	REF
V_{OH}: H-state output voltage HoC, $Ho_{7..0}$				V	VDD
V_{OL}: L-state output voltage HoC, $Ho_{7..0}$				V	VDD
V_{IH}: H-state input voltage HiC, $Hi_{7..0}$				V	VDD
V_{IL}: L-state input voltage HiC, $Hi_{7..0}$				V	VDD
I_{OH}: H-state output current HoC, $Ho_{7..0}$				mA	
I_{OL}: L-state output current HoC, $Ho_{7..0}$				mA	
I_{IH}: H-state input current HiC, $Hi_{7..0}$				mA	
I_{IL}: L-state input current HiC, $Hi_{7..0}$				mA	
C_{IN}: Input capacitance HiC, $Hi_{7..0}$				pF	
C_{OUT}: Output capacitance HoC, $Ho_{7..0}$				pF	

Switching characteristics	MIN	TYP	MAX	UNIT
t_{BC}: HiC clock cycle time	1000			ps
t_{BCH}: HiC clock high time	400			ps
t_{BCL}: HiC clock low time	400			ps
t_{BT}: HiC clock transition time			100	ps
t_{BS}: set-up time, $Hi_{7..0}$ valid to HiC xition	200		100	ps
t_{BH}: hold time, HiC xition to $Hi_{7..0}$ invalid	-200		-100	ps
t_{OS}: skew between HoC and $Ho_{7..0}$	-50		50	ps

Logical Memory Structure

Hermes defines a logical memory region as an array of 2^{8A} blocks of size W bytes. Each access, either a read or write, references all bytes of a single block. All addresses are block addresses, referencing the entire block.



Logical memory organization

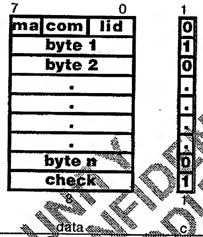
Hermes defines a logical cache for data originally contained in the logical memory region. All accesses to Hermes memory space maintain consistency between the contents of the cache and the contents of the logical memory region.

Packet Structure

Packets sent on a Hermes channel contain control commands, most commonly read or write operations, along with addresses and associated data. Other commands indicate error conditions and responses to the above commands.

When the Hermes channel is otherwise idle, such as during initialization and between packets, an idle packet, consisting of a pair of an all-zero byte and all-one byte is transmitted through the channel. Each non-idle packet consists of two bytes or a multiple of two bytes and must begin with a byte of value other than all-zero (0). All packets begin during a clock period in which the clock signal is zero, and all packets end during a clock period in which the clock signal is one.

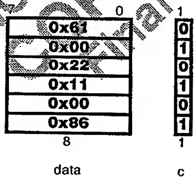
The general form of a packet is an array of bytes, without a specified byte ordering. The first byte contains a module address in the high-order two bits, a packet identifier, usually a command, in the next three bits, and a link identification number. The remaining bytes' interpretation are dependent upon the packet identifier:



General packet

The length of the packet is implied by the command specified by the initial byte of the packet.

The check byte is computed as odd bit-wise parity, with a leftward circular rotation after accumulating each byte. This algorithm provides detection of single-bit and some multiple-bit errors with high probability $(1-2^{-8})$, but no correction. As an example, the following packet has a proper check byte:



The check byte in this example is calculated as:

binary	hex	notes
01100001	61	first byte
11000010	c2	shift left circular
00000000	00	second byte
11000010	c2	xor above two rows
10000101	85	shift left circular
00100010	22	third byte
10100111	a7	xor above two rows
01001111	4f	shift left circular
00010001	11	fourth byte
01011110	5e	xor above two rows
10111100	bc	shift left circular
00000000	00	fifth byte
10111100	bc	xor above two rows
01111001	79	shift left circular
10000110	86	sixth (check) byte
11111111	ff	xor above two rows
11111111	ff	shift left circular

The general interpretation of the packet command is given in the following table:

value	interpretation	payload
0	idle	0
1	error	0
2	write-allocate	12
3	write-noallocate	12
4	read-allocate	4
5	read-noallocate	4
6	read-response	8
7	write-response	0

Packet command interpretation

The module address field provides for as many as four Hermes slave devices to be operated from a single channel. Module address values are assigned via either static/geometric configuration pins (not recommended) or dynamically assigned via a Cerberus configuration register.

The link identification field provides the opportunity for Hermes master devices to initiate as many as eight independent operations at any one time to each Hermes slave device. Each outstanding operation to a Hermes slave device must have a distinct link identification number, and no ordering of operations is implied by the value of the link-identification field. There is no requirement for link-identification field values to be sequentially assigned in requests or responses.

The following section provides detailed descriptions of the structure of each type of command packet.

Idle

Idle packets fill the space between other packets with an alternating zero-byte and all-ones-byte pattern. Idle packets may be dropped when received and regenerated between outgoing packets. The idle packet is formatted as follows:

7		0
ma	**com**	**lid**
check		

8

Idle packet

The range of valid values and the interpretation of the fields is given by the following table:

field	value	interpretation
ma	0	Module address field must be zero.
com	0	Packet is "idle."
lid	0	Link identification number field must be zero.
check	255	Check integrity of packet transmission

Idle packet field interpretation

No activity is performed upon receipt of a properly formatted idle packet.

Read Operation

Read packets cause a Hermes device to perform a read operation for the specified address, producing a data value. The value is read from cache, if one is present and the address is present in the cache. If the address is not present in cache, the value is read. A value read is placed in the cache if the command is "read-allocate"; if the command is "read-noallocate" the value is returned without copying the value into the cache. The packet format is as follows:

7		0
ma	**com**	**lid**
addr$_{7..0}$		
•		
•		
addr$_{8A-1..8A-8}$		
check		

8

Read packet

MU 0023489

Highly Confidential

The range of valid values and the interpretation of the fields is given by the following table:

field	value	interpretation
ma	0..3	Module address.
com	4, 5	Packet command is "read-allocate" or "read-noallocate."
lid	0..7	Respond with link identification number **id**.
addr	0..2^{8A}-1	Logical memory block address as specified. The least significant byte is sent first.
check	0..255	Check integrity of packet transmission.

Read packet field interpretation

If the fields are valid and the specified address is within the range of the memory, the memory is read and a read response packet is generated which contains the requested data value. The "read-response" packet is formatted as follows:

7		0
ma	**com**	**lid**
$data_{7..0}$		
.		
.		
.		
.		
.		
.		
$data_{8W-1..8W-8}$		
check		
8		

Read-response packet

MU 0023490

The range of valid values and the interpretation of the fields is given by the following table:

field	value	interpretation
ma	0..3	Module address **ma** as specified in read packet.
com	6	Packet command is "read response."
lid	0..7	Link identification number **lid** as specified in read packet.
data	$0..2^{8w}-1$	Data read from specified address.
check	0..255	Check integrity of packet transmission.

Read response packet field interpretation

In order to reduce the latency of read response, Hermes devices may generate a read response packet before checking redundant information that may alter the contents of the response. If, upon checking the information, but before the last byte of the read response packet is generated, the device detects that the data was transmitted in error, the packet is "stomped," that is, marked as invalid, by transmitting a check byte that is the ones-complement of the proper check byte. Such a packet must be ignored by Hermes masters and may be either ignored or suppressed by Hermes slave devices. If the redundant information indicates a correctable error, the stomped packet is followed by a read response packet which contains the corrected data.

Write Operation

Write packets cause Hermes devices to perform a write operation, placing a data value into the specified address. The value is written into cache, if one is present and the address is present in the cache. If the address is not present in cache, and the command is "write-allocate", the value is written into cache. If the address is not present in cache, and the command is "write-noallocate", the value is written, leaving the cache location unchanged. The packet format is as follows:

7 … 0
ma | **com** | **lid**
addr$_{7..0}$
.
.
addr$_{8A-1..8A-8}$
data$_{7..0}$
.
.
.
.
.
.
data$_{8W-1..8W-8}$
check
8

Write packet

The range of valid values and the interpretation of the fields is given by the following table:

field	value	interpretation
ma	0..3	Module address.
com	2, 3	Packet command is "write-allocate" or "write-noallocate."
lid	0..7	Respond with link identification number **lid**.
addr	0..$2^{8\mathbf{A}}$-1	Logical memory block address as specified. The least significant byte is sent first.
data	0..$2^{8\mathbf{W}}$-1	Data to be written at specified address.
check	0..255	Check integrity of packet transmission.

Write packet field interpretation

MU 0023492

If the fields are valid and the specified address is within the range of the memory, the memory is written and a write response packet is generated. The "write-response" packet is formatted as follows:

7		0
ma	com	lid
check		
8		

Write response packet

The range of valid values and the interpretation of the fields is given by the following able:

field	value	interpretation
ma	0..3	Module address **ma** as specified in write packet.
com	7	Packet command is "write response."
lid	0..7	Link identification number **lid** as specified in write packet.
check	0..255	Check integrity of packet transmission.

Write response packet field interpretation

Error Handling

The receipt of packets that do not conform to the requirements of this specification over the input channel is an error, as are any conditions internal or external to the device that prevent proper operation, such as uncorrectable memory errors. The level or degree to which an implementation detects errors is implementation-defined; to the extent possible, this architecture specification recommends that all errors should be detected, but this is not strictly required. All implementations must document the level of error detection, and all detected errors must use the method described below for handling errors.

For Hermes devices, the following errors should be detected and the level of error detection for each of these errors is required to be documented:

MU 0023493

errors detected
invalid check byte
invalid command
invalid address
uncorrectable error in cache
uncorrectable error in device
invalid identification number
internal buffer overflow
invalid module address on idle packet
invalid identification number on idle/error packet
invalid check byte on idle packet

Packets received by Hermes devices may have an invalid check byte, invalid command, invalid module address, invalid address, invalid identification number, or in some implementations cause internal buffer overflow. For each such error which the implementation may detect, the device causes a response explicitly indicating such a condition (error response); the packet is otherwise ignored. Also, detection of an uncorrectable error in either the cache or the device resulting from a request over a Hermes input channel results in the generation of an error response packet.

The error response packet is formatted as follows:

7		0
ma	com	lid
check		
8		

Error response packet

The range of valid values and the interpretation of the fields is given by the following table:

field	value	interpretation
ma	0..3	Module address identifying the source of the error response packet.
com	1	Packet is "error response."
lid	0	Link identification number must be zero.
check	0..255	Check integrity of packet transmission.

error response packet field interpretation

MU 0023494

Upon receipt of the error response packet, the packet originator must read the Cerberus status register of the reporting device to determine the precise nature of the error. Hermes devices reporting an invalid packet will suppress the receipt of additional packets until the error is cleared, by clearing the status register. However, such devices may continue to process packets which have already been received, and generate responses. Upon taking appropriate corrective actions and

clearing the error, the packet originator should then re-send any unacknowledged commands.

Because of the large difference in clock rate between the high-bandwidth Hermes channel and the Cerberus serial bus interface, it is generally safe to assume that, after detecting an error response packet, an attempt to read the status register via Cerberus will result in reading stable, quiescent error conditions and that the queue of outstanding requests will have drained. After clearing the status register via Cerberus, the packet originator may immediately resume sending requests to the Hermes device.

Forwarding

Hermes devices, whether master or slave, may have the capability to forward packets which are intended for other devices connected to a Hermes channel. For slave devices, this forwarding is performed on the basis of the contents of the module address field in the packet; packets which contain a module address other than that of the current device are forwarded. All non-idle packets which contain such module addresses must be forwarded, including error packets. For master devices, this forwarding is performed on the basis of the identifier number field in the packet; packets which contain identifier numbers not generated by the device are forwarded.

To minimize ring latency, it is generally desirable to forward these packets with minimal latency. If a packet arrives at an input channel when the output channel is in use, this latency must increase; at least a single packet buffer is required.

The size of the forwarding buffer is implementation-dependent. Avoiding the generation of an output packet if the forwarding buffer does not have room to hold an additional input packet is required, when the forwarding buffer is smaller than the number of packets which may require forwarding (generally 24 packets). However, this strategy may cause starvation, as output packets may be inhibited indefinately by a stream of input packets that require forwarding. Starvation may be avoided by system-level design and configuration considerations beyond the scope of this specification.

Packets which contain a check byte error may be forwarded; however it is recommended that such packets be transmitted with a check byte containing more than one error bit, to minimize the possibility of an undetected second error. Packets which contain a "stomped" check byte may be forwarded as is, or may be ignored by a forwarding device. Note that when a packet is forwarded with minimum latency, the output channel may begin transmitting a packet before the input channel has received the entire packet: in such a case, the only available choice is to continue forwarding the packet even if a check byte error or "stomped" check byte is detected.

Ring Configurations

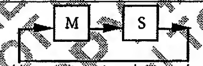
Hermes supports a variety of ring configurations. All devices in a cascade must have the same values for A and W parameters, in order that each part may properly interpret packet boundaries. The table below summarizes the characteristics of the configurations available:

configuration	masters		slaves	
	number	forwarding	number	forwarding
master-slave pair	1	no	1	no
single-master ring	1	no	1-4	yes
dual-master pair	2	no	0	
multiple-master single-slave ring	1-8	yes	1	no
multiple-master multiple-slave ring	1-8	yes	1-4	yes

Hermes ring configurations

Master-slave Pair

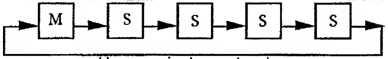
The simplest ring consists of a single Hermes non-forwarding master device and a single Hermes non-forwarding slave device. No forwarding is required for either device as packets are sent directly to the recipient. The ring may have as many as eight transactions outstanding, each containing distinct id field values.



Hermes master-slave pair

Single-master Ring

A single-master ring may contain a cascade of up to four Hermes slave devices. The cascade of devices will have the same or greater bandwidth as a single device, but more latency. Each Hermes slave device must be configured to a distinct module address, and each slave device must forward packets that contain module address fields unequal to their own.



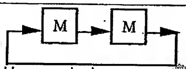
Hermes single-master ring

Packets are explicitly addressed to a particular Hermes device; any packet received on a device's input channel which specifies another module address is automatically passed on via its output channel. This mechanism provides for the serial interconnection of Hermes devices into strings, which function identically to a single device, except that a string has larger capacity and longer response

latency. Each slave device may have as many as eight transactions outstanding, each containing distinct **id** field values.

Dual-master Pair

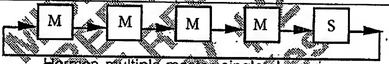
A dual master pair consists of two master devices and no slave devices. Each master device may initiate read and write operations addressed to the other, and each may have up to eight such transactions outstanding. No forwarding is required for either device as packets are sent directly to the recipient.



Hermes dual-master pair

Multiple-master Single-slave Ring

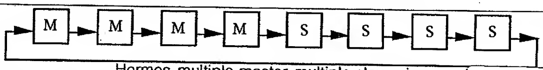
A multiple-master ring may contain multiple master devices and a single Hermes slave device, provided that the master devices arrange to use different **id** values for their requests. Each master may use a share of the eight transactions. Master devices must forward packets not specifally addressed to them, as designated by the values in the **id** field. The slave device need not forward packets, as all input packets are designated for the slave device.



Hermes multiple-master single-slave ring

Multiple-master Multiple-slave Ring

A multiple-master ring may contain multiple master devices and as many as four Hermes slave devices, provided that the master devices arrange to use different **id** values for their requests. Each slave may have up to eight transactions outstanding, and each master may use a share of those transactions. Master devices must forward packets not specifally addressed to them, as designated by the values in the **id** field. Slave devices must forward packets not specifically addressed to them, as designated by the value of the **ma** field.



Hermes multiple-master multiple-slave ring

MU 0023497

Response Packet Timing

In general, a received packet which is interpreted as a command causes a response packet to be generated. The latency between the end of the request packet and the beginning of the response packet is affected by the processing and forwarding of other packets, by the presence or absence of the requested word in

the cache, as well as implementation-dependent device parameters and characteristics.

With full knowledge of the cache state, configurable parameters and implementation-dependent characteristics, a Hermes master may completely model the latency of responses. However, dependence on such characteristics is not recommended, except for testing and characterization purposes.

A Hermes master must have the capability to detect a time-out condition, where a response to a request packet is never received. The length of the time-out is implementation-defined, and dependent upon the implementation of the Hermes slave devices, so it is recommended that this time-out be long enough to accomodate variation in the design of Hermes slave devices, or be configurable to permit recovery in a minimum implementation-dependent delay.

Cerberus Registers

The Hermes channel architecture builds upon the Cerberus serial bus architecture. Only the specific requirements of Hermes-compliant devices are defined below.

Hermes requires that the values of A and $\log_2 W$ be made available in the high-order byte of the first architecture description register as indicated below.

The format of the register is described in the table below. The **octlet** is the Cerberus address of the register; **bits** indicate the position of the field in a register. The **value** indicated is the hard-wired value in the register for a read/only register, and is the value to which the register is initialized upon a reset for a read/write register. If a reset does not initialize the field to a value, or if initialization is not required by this specification, a * is placed in or appended to the value field. The **range** is the set of legal values to which a read/write register may be set. The **interpretation** is a brief description of the meaning or utility of the register field; a more comprehensive description follows this table.

octlet	bits	field name	value	range	interpretation
4	63..60	**A**	4	1..15	size of a Hermes address
	59..56	**$\log_2$W**	3	0..15	size of a Hermes word
	55..0	**not specified**			not specified by Hermes architecture

Architecture Description Registers

The architecture description registers in octlets 4 and 5 comply with the Cerberus specification and contain a machine-readable version of the architecture parameters: A, W described in this document.

MU 0023498

Cerberus Serial Bus

MicroUnity's Cerberus serial bus architecture is designed to provide bootstrap resources, configuration and diagnostic support for MicroUnity's Terpsichore system architecture.

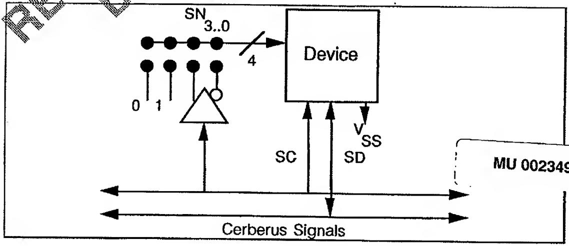
The Cerberus serial bus employs two signals, both at TTL levels, for direct communication among as many as 2^8 devices. One signal is a continuously running clock, and the other is an open-collector bidirectional data signal. Four additional signals provide a geographic 8-bit address for each device. A gateway protocol and optional configurable addressing each provide a means to extend Cerberus to as many as 2^{16} buses and 2^{24} devices.

The protocol is designed for universal application among the custom chips used to implement the Terpsichore system architecture. It is also designed to be compatible with implementations embodied in FPGA parts, such as those made by Xilinx, Altera, Actel and others. Such FPGA parts may be used to adapt the Cerberus protocol in a minimum of logic to attach small serial bus devices, such as those made by Dallas Semiconductor (EEPROM, serial number parts), ITT (IMB bus), Signetics (I^2C bus). It is also a goal that such FPGA parts can be used to adapt the Cerberus protocol to communication over EIA 232/422/423/485 links to existing systems for the purposes of system development, manufacturing test and configuration, and manufacturing rework.

The Cerberus serial bus is used for the initial bootstrap program load of Terpsichore; the bootstrap ROM connects to Terpsichore via Cerberus. Because the Cerberus must be operational for the fetch of the first instruction of Terpsichore, the bus protocol has been devised so that no transactions are required for initial bus configuration or bus address assignment.

Electrical Signalling

The diagram below shows the signals used in Cerberus.



The SC signal is a continuously running clock signal at TTL levels. The rate is specified as 20 MHz maximum, 0 (DC) minimum. The SC signal is sourced from a single point or device, possibly through a fan-out tree, the location of which is unspecified.

The actual clock rate used is a function of the length of the bus and quality of the noise and signal termination environment. The amount of skew in the SC signal between any two Cerberus devices should be limited by design to be less than the skew on the SD signal.

The SD signal is a non-inverted open-collector (0 = driven = low; 1 = released = high) bidirectional data signal, at TTL levels, used for all communication among devices on Cerberus.

One of several termination networks may be used on this signal, depending upon joint design targets of network size, clock rate, and cost. One of the simplest schemes employs a resistive pull-up of the equivalent of 220 Ohms to 3.3 Volts above V_{SS}. A more complex termination network, such as termination networks including diodes, or the "Forced Perfect Termination" network proposed for the SCSI-2 standard may be advantageous for larger configurations. Termination voltages as high as 3.3 V are permitted.

The following table specifies parameters that must be met by Cerberus-compliant devices. Voltages are referenced to V_{SS}.

Recommended operating conditions	MIN	NOM	MAX	UNIT
Operating free-air temperature	0		70	C

Electrical characteristics	MIN	TYP	MAX	UNIT
V_{OL}: L-state output voltage	0		0.5	V
V_{IH}: H-state input voltage SD	2.0		V_T+0.5	V
V_{IH}: H-state input voltage SC, $SN_{3..0}$	2.0		5.5[47]	V
V_{IL}: L-state input voltage	-0.5		0.8	V
I_{OL}: L-state output current[48]			16	mA
I_{OZ}: Off-state output current[49]	-10		10	µA
C_{OUT}: Output Capacitance			4.0	pF

Switching characteristics	MIN	TYP	MAX	UNIT
t_C: SC clock cycle time	50			ns
t_{CH}: SC clock high time	20			ns
t_{CL}: SC clock low time	20			ns
t_T: SC clock transition time			5	ns
t_S: set-up time, SD valid to SC rise	0			ns
t_H: hold time, SC rise to SD invalid			1	ns
t_{OD}: SC rise to SD valid	5			ns

Geographic addressing

The objective of the geographic addressing method in Cerberus is to ensure that each device is addressable with a number which is unique among all devices on the bus and which reflects the physical location of the device, so that the address remains the same each time the system is operated.

When a system requires at most 16 devices, the geographic addressing method permits the assignment of addresses 0 through 15 by directly wiring the low-order 4 bits of the address in binary code using input signals $SN_{3..0}$. For these purposes, wiring to a logic high (H) level supplies a value of 1, and wiring to V_{SS} or logic low (L) level supplies a value of 0.

47 Cerberus recommends, but not require, compliant devices be able to sustain input levels provided by 5V TTL-compatible devices on the SC and $SN_{3..0}$ inputs.

48 Devices which fail to comply with the low-state output current specification may operate with Cerberus-compliant devices, but may require changes to the termination network. System designers should evaluate the effect that limited drive current will have on the worst-case Low-state signal level.

49 Devices which fail to comply with the off-state output current specification may operate with Cerberus-compliant devices, but may limit the number of devices which may co-exist on a single Cerberus bus. System designers should evaluate the effect that additional leakage current will have on the worst-case High-state and Low-state signal levels.

The table below indicates the wiring pattern for each device address from 0 through 15:

Device address	Binary code	SN_3	SN_2	SN_1	SN_0
0	00000000	L	L	L	L
1	00000001	L	L	L	H
2	00000010	L	L	H	L
3	00000011	L	L	H	H
4	00000100	L	H	L	L
5	00000101	L	H	L	H
6	00000110	L	H	H	L
7	00000111	L	H	H	H
8	00001000	H	L	L	L
9	00001001	H	L	L	H
10	00001010	H	L	H	L
11	00001011	H	L	H	H
12	00001100	H	H	L	L
13	00001101	H	H	L	H
14	00001110	H	H	H	L
15	00001111	H	H	H	H

An extension of this method is used for the assignment of addresses 0 through 255 when a system requires more than 16 devices, up to 2^8 devices. Additional code combinations are made available by wiring each of the same input signals $SN_{3..0}$ as before to one of four signals: the two described above, L and H, and two additional signals, a buffered copy of the SC signal and an inverted copy of the SC signal (SC_N). Since there are four SN signals, each wired to one of four values, $4^4=2^8=256$ combinations are possible.

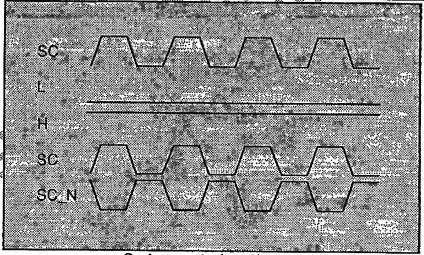
The wiring pattern is constructed using the algorithm: If the desired device address is the value N, for each input signal SN_x, where x is in the range 3..0, wire SN_x to one of the four signals L, H, SC, or SC_N, according to the following table, depending on the value of bit 4+x and bit x of N.

N_{4+x}	N_x	SN_x
0	0	L
0	1	H
1	0	SC
1	1	SC_N

The table below indicates the wiring pattern for some device addresses:

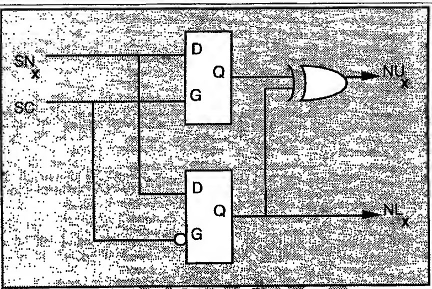
Device address	Binary code	SN_3	SN_2	SN_1	SN_0
16	00010000	L	L	L	SC
17	00010001	L	L	L	SC_N
18	00010010	L	L	H	SC
19	00010011	L	L	H	SC_N
...					
29	00011101	H	H	L	SC_N
30	00011110	H	H	H	SC
31	00011111	H	H	H	SC_N
32	00100000	L	L	SC	L
33	00100001	L	L	SC	H
34	00100010	L	L	SC_N	L
...					
254	11111110	SC_N	SC_N	SC_N	SC
255	11111111	SC_N	SC_N	SC_N	SC_N

The diagram below shows the waveform of the SC signal and the four signals that each of the $SN_{3..0}$ inputs may be wired to.



Cerberus device Signals

The values shown in the diagram above are decoded using four copies of the following logic, one for each value of x in the range 3..0:



Cerberus Selection Decoding

The NU and NL values are combined together in the order:

7	6	5	4	3	2	1	0
NU_3	NU_2	NU_1	NU_0	NL_3	NL_2	NL_1	NL_0
1	1	1	1	1	1	1	1

to construct an 8-bit device number by which operations are addressed.

Bit-Level Protocol

MU 0023504

The communication protocol rests upon a basic mechanism by which any device may transmit one bit of information on the bus, which is received by all devices on the bus at once. Implicit in this mechanism is the resolution of collisions between devices which may transmit at the same time.

Each transmitted bit begins at the rising edge of the SC signal, and ends at the next rising edge. The bit value is sampled by all devices at the next rising edge of the SC signal, thus permitting relatively large signal settling time on the SD signal, provided that skew on the SC signal is adequately controlled.

The transmission of a zero (0) bit value on the bus is performed by the transmitter driving the SD signal to a logical-low value. The transmission of a one (1) bit value on the bus is performed by the transmitter releasing the SD signal to attain a logical-high value (driven by the signal termination network). If more than one device attempts to transmit a value on the same clock period (of the SC signal), the resulting value is a zero if any device transmits a zero value, and is a one if all devices transmit a one value. We define the occurrence of one or more devices transmitting a zero value on the same clock cycle where one or more devices transmit a one value as a collision.

Highly Confidential

Because of this wired-and collision mechanism, if a device transmits a zero value, it cannot determine whether any other devices are transmitting at the same time. If a device transmits a one value, it can monitor the resulting value on the SD signal to determine whether any other device is transmitting a zero value on the same clock cycle. In either case, if two or more devices transmit the same value on the same clock cycle, neither device, in fact, no device on the bus can detect the occurrence, and we do not define such an occurrence as a collision.

This collision mechanism carries over to the higher levels of the protocol, where if two or more devices transmit the same packet or carry on the same transaction, no collision occurs. In such cases, the protocol is designed so that the transaction occurs normally. These transactions may occur frequently if two identical devices are reset at the same time and each initiates bus transactions, such as two processors each fetching bootstrap code from a single shared ROM device.

Packet Protocol

The packet protocol uses the bit-level mechanism to transmit information on the bus in units of eight bits or a multiple of eight bits, while resolving potential collisions between devices which may simultaneously begin transmitting a packet. The transmission provides for the detection of single-bit transmission errors, and for controlling the rate of information flow, with eight-bit granularity. The protocol also provides for the transmission of a system-level reset.

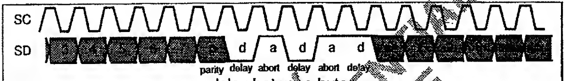
Each packet transmission begins with a single start bit, in which SD always has a zero (driven) value. Then the bits of the first data byte are serially transmitted, starting with the least-significant bit. After transmitting the eight data bits, a parity bit is transmitted. If transmission continues with additional data, a single one (released) bit is transmitted, immediately followed by the least-significant bit of the next byte, as shown in the figure below.

Packet transmission

MU 0023505

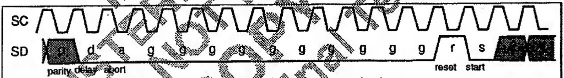
Highly Confidential

Otherwise, on the cycle following the transmission of each parity bit, any device may demand an additional delay of two cycles to process the data by driving the SD signal (to a zero value) and then, on the next cycle releasing the SD signal (to a one value), making sure that the signal was not driven (to a zero value) by any other device. Further delays are available by repeating the pattern of driving the SD signal (to a zero value) for one cycle and releasing the SD signal (to a one value) for one cycle, and ensuring that the signal has been released. Additional bytes are transmitted immediately after the bus has been one (released) on the "d" (delay) clock cycle, without additional start bits, as shown in the figure below.



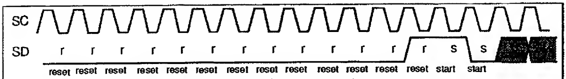
delay between bytes

Any Cerberus device may abort a transaction, usually because of a detected parity error or a deadlock condition in a gateway, by driving the SD signal (to a zero value) on the "d" (delay) and the "a" (abort) cycles, as well as the next ten cycles, for a total of 12 cycles. The additional ten cycles ensure that the abort is detected by all devices, even under the adverse condition where a single-bit transmission error has placed devices into inconsistent states. Each device that detects an abort drives the SD signal (to a zero value) for ten cycles after its "a" (abort) cycle state, so in the most adverse case, an abort may have devices driving the bus to as many as 22 consecutive cycles. The figure below shows a typical (12 cycle) transaction abort, followed by an immediate re-transmission of the transaction.



abort followed by re-transmission

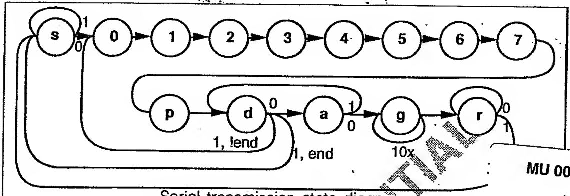
Any Cerberus device may reset the Cerberus bus and all Cerberus devices, by driving the SD signal (to a zero value) for at least 33 cycles. This is sufficient to ensure that all devices receive the reset no matter what state the device is in prior to the reset. Transmission may resume after the SD signal is released (to a one value) for two cycles, as shown in the figure below.



reset followed by transmission

MU 0023506

The state diagram below describes this protocol in further detail:



Serial transmission state diagram

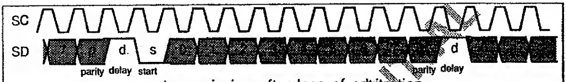
The table below describes the data output and actions which take place at each state in the above diagram. The next state for each state in the table is either column go-0 or go-1, depending on the value of the in column.

state	out	in	go-0	go-1	action
s	s	i_s	0	s	s = 0 iff transmit first byte. Must wait in this state one cycle (with s=1) if transmitting a new transaction.
0	d_0	i_0	1	1	bit 0 (LSB) of data. If d_0 &~ i_0, lose arbitration.
1	d_1	i_1	2	2	bit 1 of data. If d_1 &~ i_1, lose arbitration.
2	d_2	i_2	3	3	bit 2 of data. If d_2 &~ i_2, lose arbitration.
3	d_3	i_3	4	4	bit 3 of data. If d_3 &~ i_3, lose arbitration.
4	d_4	i_4	5	5	bit 4 of data. If d_4 &~ i_4, lose arbitration.
5	d_5	i_5	6	6	bit 5 of data. If d_5 &~ i_5, lose arbitration.
6	d_6	i_6	7	7	bit 6 of data. If d_6 &~ i_6, lose arbitration.
7	d_7	i_7	p	p	bit 7 (MSB) of data.. If d_7 &~ i_7, lose arbitration.
p	p	i_p	d	d	p = ~^$i_{7..0}$ (odd parity); abort if p^i_p.
d	d	i_d	a	s/0	d = 0 iff transmit delay, abort, or reset. If id=1, go to state 0 if not last byte of packet; else state s.
a	a	i_a	g	d	a = 0 iff transmit abort or reset. If i_a = 0, abort transaction.
g	0	N/A	g/r	N/A	stay in state g 10 times, then go to state r.
r	r	i_r	r	s	r = 0 iff transmit reset. If i_r = 0 and have been in this state 12 times, reset device.

In order to avoid collisions, no device is permitted to start the transmission of a packet unless no current transaction is underway. To resolve collisions that may occur if two devices begin transmission on the same cycle, each transmitting device must monitor the bus during the transmission of one (released) bits. If any

of the bits of the byte are received as zero (driven) when transmitting a one (released), the device has lost arbitration, and must transmit no additional bits of the current byte or transaction.

A device which has lost the arbitration of a collision, or has suffered the occurrence of a transaction abort, may retry the transmission immediately after the transmission of the last byte of the current transaction, as shown in the figure below.



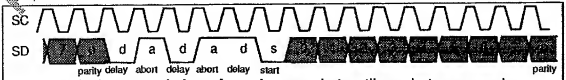
re-transmission after loss of arbitration

All other devices must wait one additional SC (clock) cycle before transmitting another message, as shown in the figure below. This ensures that all devices which have collided perform their operations before another set of devices arbitrate again.

re-transmission after loss of arbitration

All initiator-capable devices must enforce a time-out limit of no more than 256 idle clock cycles between the packets of a transaction. After seeing this many idle clock cycles, at some time within the next 256 clock cycles, such devices must abort the current transaction transmitting a time-out packet, which consists of two bytes of zeroes.

Slow devices may require more cycles between the transmission of packets in a transaction than are permitted as idle clock cycles. Such devices may avoid the time-out limit by delaying the completion of the transmission of the previous packet until the idle time is less than the time-out limit, as shown in the figure below. In this way, devices of any speed may be accommodated.



delaying completion of previous packet until ready to respond

It is necessary that initiator-capable and other devices cooperatively avoid collisions between the time-out packet and transaction responses. The responsibility of the initiator devices is to inhibit transmission of a time-out packet if, before the time-out packet can be transmitted, some device begins transmitting, even if such a transmission begins after 256 idle clock cycles have elapsed. If the design of a target device ensures that no more than 256 idle clock cycles elapse between packets of a transaction, it need not be concerned of the possibility of a

collision during the transmission of a response packet. Otherwise, the responsibility of the target devices is to inhibit transmission of a response if some other device begins transmitting a time-out packet after at least 256 idle clock cycles have elapsed.

A device which requires delay after an aborted transaction or a reset may cause such a delay by forcing the delay bit after the first byte of the immediately following transaction, as required. If in such a case, the device cannot keep a copy of the first byte of the transaction, it may force the transaction initiator to retransmit the byte by aborting that following transaction after a suitable delay has been requested.

Transaction Protocol

A transaction consists of the transmission of a series of packets. The transaction begins with a transmission by the transaction initiator, which specifies the target net, device, length, type, and payload of the transaction, request. If the type of the packet is in the range 128..255, the target device responds with an additional packet, which contains a length and type code and payload. The transaction terminates with a packet with a type field in the range 0..127, otherwise the transaction continues with packet transmission alternating between transaction initiator and the specified target.

The general form of an initial packet is:

h_0	n_1	de	L	T	p_0	p_1	....	p_{L-1}

The general form of subsequent packets is:

L	T	p_0	p_1	....	p_{L-1}

MU 0023509

Highly Confidential

The range of valid values and the interpretation of the bytes is given by the following table:

Field	Value	Interpretation
n_0, n_1	$0..2^{16}-1$	network address of target, relative to network address of transaction initiator. Value is zero (0) if target is on same bus as transaction initiator.
de	0..255	device address, in this case, an absolute value, *i.e.*, not relative to device address of transaction initiator.
L	0..255	payload length, or number of bytes after transaction code (**T**).
T	0..255	transaction code. If the transaction code is in the range of 0..127, the transaction is terminated with this packet. If the transaction code is in the range of 128..255, the transaction continues with additional packets.
$p_0,p_1,...p_{L-1}$	0..255	Payload of transaction.

general transaction byte interpretation

The valid transaction codes are given by the following table:

mnemonic	L	T	interpretation
te	0	0	transaction error: bus timeout, invalid transaction code, invalid address.
tc	0	1	transaction complete: normal response to a write operation
d8	8	2	data returned from read octlet
		3..127	reserved for future definition
w8	10	128	write octlet
r8	2	129	read octlet
		130..255	reserved for future definition

general transaction byte interpretation

MU 0023510

All Cerberus devices must support the transaction codes: te, tc, d8, w8, and r8.

All Cerberus devices monitor SD to determine when transactions begin and end. A transaction is terminated by the completion of the transmission of the specified number of payload bytes in a transaction with code in the range 128..255, or by the transmission of an abort sequence. For purposes of monitoring transaction boundaries, only the L byte is interpreted; the value of the T byte (except for the high order bit) must be disregarded. This is of particular importance as many transaction codes are reserved for future definition. and the use of such

transaction codes between devices which support them must be permitted, even though other devices on the Cerberus bus may not be aware of the meaning of such transactions. A Cerberus device must permit any value in the L byte for transactions addressed to other devices, even if only a limited set of values is permitted for transactions addressed to that device.

Transactions addressed to a device which does not provide support for the enclosed transaction code or payload length should be aborted by the addressed target device.

The selection of the payload length L and transaction code T for the transaction error packet is of particular note. Because the value of all information bits of the packet is zero, it is guaranteed that a device which transmits this packet will have collision priority over all others.

Write Octlet

The "write octlet" transaction causes eight bytes of data to be transferred from the transaction initiator to the addressed target device at an octlet-aligned 16-bit device address. The transaction begins with a request packet of the form:

n_0	n_1	de	10	w8	A_0	A_1	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7

The normal response to this request is of the form:

0	tc

The error response to this request is of the form:

0	te

The 16-bit device address is interpreted as an octlet address (not a byte address) and is assembled from the A_0 and A_1 bytes as (most significant byte is transmitted first):

15 8	7 0
A_0	A_1
8	8

The data to be transferred to the target device is assembled into an octlet as (most significant byte is transmitted first):

63 56	55 48	47 40	39 32	31 24	23 16	15 8	7 0
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
8	8	8	8	8	8	8	8

Side-effects due to the alteration of the contents of the octlet at the specified address are only permitted if the transaction completes normally. In the event that the write octlet transaction is aborted at or prior to the transmission of the A_1 byte, the target device must make no permanent state changes. If the transaction

is aborted at or after the transmission of the D_0 byte, the contents of the octlet at the specified address is undefined. If alterations of the contents normally would cause side-effects in the operation of the Cerberus device or side-effects on the contents of other addressable octlets in the device, these side-effects must be suppressed.

If the addressed target device is not present on the Cerberus bus, the transaction will proceed to the point of transmitting the octlet data and then stop until the idle time-out limit is reached. At that point, one or more initiator-capable devices will generate an error response packet.

If the addressed target device is present on the Cerberus bus, but the 16-bit device address is not valid for that device, the target must generate an error response packet.

Read Octlet

The "read octlet" transaction causes eight bytes of data to be transferred to the transaction initiator from the addressed target device at an octlet-aligned 16-bit device address. The transaction begins with a request packet of the form:

n_0	n_1	de	2	r8	A_0	A_1

The normal response to this request is of the form:

8	d8	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7

The error response to this request is of the form:

0	te

The 16-bit device address is interpreted as an octlet address (not a byte address) and is assembled from the A_0 and A_1 bytes as (most significant byte is transmitted first):

15 8	7 0
A_0	A_1
8	8

MU 0023512

The data to be transferred to the target device is assembled into an octlet as (most significant byte is transmitted first):

63 56	55 48	47 40	39 32	31 24	23 16	15 8	7 0
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
8	8	8	8	8	8	8	8

Regardless of whether the transaction completes, the read octlet transaction must have no side-effects on the operation of the Cerberus device or the contents of other addressable octlets.

If the addressed target device is not present on the Cerberus bus, the transaction will proceed to the point of transmitting the octlet address and then stop until the idle time-out limit is reached. At that point, one or more initiator-capable devices will generate an error response packet.

If the addressed target device is present on the Cerberus bus, but the 16-bit device address is not valid for that device, the target must generate an error response packet.

Dedicated Octlets

Certain octlet addresses are assigned by which all Cerberus devices may be identified as to device type, manufacturer, revision, and by which devices may be individually reset and tested. All or part of octlet addresses 0..7 are reserved for this purpose.

octlet	63 56 55 48 47 40 39 32 31 24 23 16 15 8 7 0
0	identify architecture
1	identify implementation
2	identify manufacturer
3	identify serial number
4..5	identify architectural features and options
6	specify operating modes
7	report operating status
8..2^{16}-1	not specified by Cerberus
	8 8 8 8 8 8 8 8

The octlets at addresses 0 through 3 identifies the company which specifies the device architecture (*e.g.* MicroUnity), the device architecture (*e.g.* Mnemosyne, Terpsichore, Calliope), the implementor (*e.g.* MicroUnity, partner), the device implementation and manufacturer and manufacturing version (*e.g.* 1.0,1.1,2.0), and optionally a unique device serial number. Addresses 0 through 2 are read/only; an attempt to write to these addresses may cause either a normal termination or an error response. Address 3 may be read/only or read/write.

octlet	63 … 16	15 … 0
0	architecture code	architecture revision
1	implementor code	implementor revision
2	manufacturer code	manufacturer revision
3	serial number	configurable address
	48	16

The octlet at address 0 contains an architecture code and revision identifier. The architecture code and revision identifies each distinctly designed architecture version of a device. Normally, a change in the upper byte of the revision indicates a change in which features may have changed. A change in the lower byte of the revision signifies a change made to repair design defects or upward-compatible revisions.

The architecture code is a unique 48-bit identifier, comprised of the concatenation of a 24-bit unique company identifier[50], and a 24-bit value specified by the designated company. This code must not duplicate 48-bit identifiers specified for this purpose, or for other purposes, including use of unique identifiers for implementation codes, manufacturing codes, or in IEEE 1212, or IEEE 802. IEEE 802 48-bit identifiers are specified in terms of a binary ordering of bits on a single line; for Cerberus, the ordering which is appropriate is that labelled "CSMA/CD and Token Bus," where bits are driven onto Cerberus with the least-significant bit of each byte first.

MicroUnity's architecture codes are specified by the following table:

Internal code name	Code number
Mnemosyne	0x00 40 a3 49 d2 a4
Euterpe	0x00 40 a3 24 69 93
Calliope	0x00 40 a3 92 b4 49

Refer to the designated architecture specification for architecture revision codes.

[50]Company identifiers are a 24-bit value assigned by authority of the IEEE. Ask for a 'unique company identifier' for your organization:

Registration Authority for Company Identifiers
The Institute of Electrical and Electronic Engineers
445 Hoes Lane
Piscataway, NJ 08855-1331
USA
(908) 562-3812

MicroUnity's unique company identifier is: 0000 0000 0000 0010 1100 0101. Only MicroUnity may assign unique 48-bit identifiers that begin with this value. Others may assign 48-bit identifiers that begin with a 24-bit company identifier assigned by authority of the IEEE.

MicroUnity will, upon request, supply unique 48-bit identifiers for architectures, implementors, or manufacturers of designs which are fully compliant with the Cerberus Serial Bus Architecture. For assignment of identifiers, contact MicroUnity:

Craig Hansen, Chief Architect
Registration Authority for Unique Identifiers
MicroUnity Systems Engineering, Inc.
255 Caspian Drive
Sunnyvale, CA 94089-1015
Tel: (408) 734-8100
Fax: (408) 734-8136

The octlet at address 1 contains an implementation code and revision identifier. The implementation code and revision identifies each distinctly designed engineering version of a device. The implementation code is a unique 48-bit identifier, as for architecture codes. Normally, a change in the upper byte of the revision indicates a change in which features may have changed, or in which all mask layers of a device have been modified. A change in the lower byte of the revision signifies a change made to repair design defects or in which only some mask layers of a device have been modified.

Refer to the designated architecture specification for the values of the implementation code and revision fields.

The octlet at address 2 contains a manufacturer code and revision identifier. The manufacturer code and revision identifies each distinct manufacturing database of an implementation. The manufacturer code is a unique 48-bit identifier, as for architecture codes. Changes in the manufacturer revision may result from modifications made to any or all mask layers to enhance yield or improve expected device performance.

Refer to the designated architecture specification for the values of the manufacturer code and revision fields.

The octlet at address 3 optionally contains a unique device serial number or random number and optionally contains a configurable address register. If the octlet does not contain a serial or random number, it must contain a 64-bit zero value.

If the octlet contains a unique device serial number, it must be a unique 48-bit value, as for architecture codes.

If the octlet contains a random number, it must be a value chosen from a uniform distribution, selected whenever the device is reset.

The optional configurable address register permits a system design in which some devices are set to identical Cerberus device addresses at system reset time, and dynamically have their addresses moved to unique addresses by some Cerberus device. The configurable address register must be set to the address designated by the $SN_{3..0}$ pins whenever the device is reset. A device which implements the configurable address capability must also implement either a unique device serial number or a random number, must implement the arbitration mechanism during responses from read-octlet requests, and must ensure that all devices which are originally set to the same address at reset time respond to a read-octlet with identical latency. An initiator device on Cerberus may set the configurable address register by reading the entire octlet at address 3, reading both the serial/random number and the configurable address register. By the use of the bitwise arbitration mechanism, only one device completes the read-octlet response packet. Then, the initiator device writes a value to octlet address 3, where the first 48 bits of the value written must match the value just read. All target devices then examine the first 48 bits of the value written, and only if the value matches the

contents of the serial/random number on the device, uses the last 16 bits[51] to load into the configurable address register. The initiator will repeat this process until there are no more devices at the original/reset address, at which time a bus time-out occurs on the read-octlet transaction.

The octlets at addresses 4 and 5 contain architecture parameters. Values are device-architecture-dependent and implementor-independent; refer to the designated architecture specification for information. Addresses 4 and 5 are read/only; an attempt to write to these addresses may cause either a normal termination or an error response.

octlet	63 ... 0
4	architecture parameters not specified by Cerberus
5	architecture parameters not specified by Cerberus
	64

Octlet 6 designates overall device settings: Values in address 6 are changed only by external devices and not by the device itself; this register is read/write. Two bits of the first byte have standard meaning for all Cerberus devices. Bits 61..0 are not specified by Cerberus except by the restriction that these values are changed only by external devices, not by the device itself; refer to the designated architecture specification for information.

Writing a one to bit 63, r, of octlet 6 causes the device to perform a device circuit reset, which is equivalent to the reset performed by driving the SD signal (to a zero value) for 33 or more cycles, and sets the device to an initial state in which previous device state may be lost, previous control settings may be lost and variable power settings are set to a minimal, functional value, after which bits 63 and 62 of the status register below are set (to ones).

Writing a one to bit 62, c, of octlet 6 causes the device to perform a device logic clear, which initializes the device to a known, quiescent, initial state, in which previous device state may be lost, but does not affect control register settings related to variable power settings, after which bits 63 and 62 of the status register below are set (to ones).

Writing a one to bit 61, s, of octlet 6 causes the device to perform a self-test, after which previous device state may be lost, and after which bit 62 of the status register below is set (to one) if the self-test yields satisfactory results. Bit 63 of the status register below is set (to one) at the end of the self-test.

octlet	63	62	61	60 ... 0
6	r	c	s	other device settings not specified by Cerberus
	1	1	1	61

[51]A 16-bit field provides for the possibility of configuring devices which respond to addresses directly that have net numbers set, thereby blurring the dividing line between Cerberus net addresses and device addresses. Gateway designers might want to consider this possibility.

Octlet 7 designates device status. Values in address 7 are normally modified only by the device itself, except when an external device may clear status or error conditions; this register is read/write. However, the only valid data which can be written to this register is a zero value, which clears any outstanding status or error reports. Two bits of the first byte have standard meaning for all Cerberus devices. Bits 61..0 are not specified by Cerberus except by the restriction that these values are modified only by the device itself except for clearing by an external device; refer to the designated architecture specification for information.

Bit 63, c, of octlet 7 indicates whether the device has completed reset, clear, or self-test.

Bit 62, s, of octlet 7 indicates whether the device has successfully completed reset, clear, or self-test.

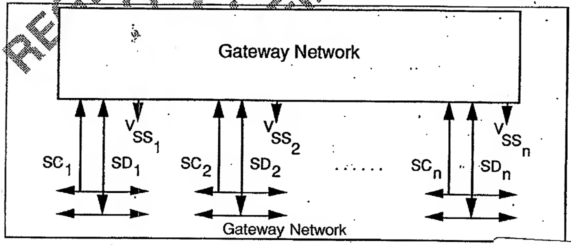
octlet	63	62	61 … 0
7	c	s	other device status not specified by Cerberus
	1	1	62

Octlets at addresses $8..2^{16}-1$ are not specified by Cerberus. Refer to the designated architecture specification for information.

Gateways

The Cerberus bus may be extended into a network of buses using a gateway. Gateways connect between buses that use the wired-and signalling protocol described above. A gateway attaches to a local Cerberus bus and receives and retransmits bus requests and responses over a linkage to other gateways, thereby reaching to additional Cerberus buses. This document does not specify the protocol used to link gateways.

The diagram below shows a gateway network connecting several Cerberus buses:



MU 0023517

Highly Confidential

Each Cerberus bus in a Cerberus network may, for specification purposes, be assigned a unique network number, in the range $0..2^{16}-1$. These network numbers never appear directly in Cerberus device addresses, as the target network byte specified in the request packet of a Cerberus transaction contains only a relative net number: the target net either minus, or xor'ed with, the initiator net. Thus, the relative target network address is always zero when the initiator and the target are on the same Cerberus bus, and is always non-zero when they are on different buses.

A Cerberus bus permits only one transaction to occur at a time. However, a Cerberus network may have multiple simultaneous transactions, so long as the target and initiator network addresses are all disjoint. In more precise terms, the network addresses must satisfy the relations:

$$\text{target}_i \neq \text{initiator}_j$$
$$\text{target}_i \neq \text{target}_j$$
$$\text{initiator}_i \neq \text{initiator}_j, \text{ for all } i \neq j.$$

A Cerberus network may set more restrictive conditions for simultaneous transactions by its internal design, as required by limits of performance or bandwidth of the gateway network. When these conditions are not satisfied, one or more transactions may be selected to be aborted on the local Cerberus bus on which they are initiated by any fair-scheduling mechanism.

Each local Cerberus bus is connected to the gateway network by exactly one gateway. When a request packet of a transaction is received by a gateway on a local Cerberus bus, the first byte of the packet specifies a net number. If this byte is non-zero, the gateway, which we will designate the initiator gateway, must carry this transaction across the gateway network. This number is interpreted as a signed byte, relative to the initiator gateway, and specifies a gateway to be the target of the transaction, which we will designate the target gateway. We will refer to the local Cerberus bus to which the initiator gateway is attached as the initiator bus, and the bus to which the target gateway is attacked as the target bus.

The request packet is carried via the initiator gateway, through the gateway network, to the target gateway, which then re-transmits the packet on the target bus. When the request packet is re-transmitted on the target bus, the network number byte is zero, designating a target on the target bus. The initiator gateway may delay transmission of the request packet on the initiator bus as required to limit or manage the flow of information through the gateway network, between each byte of the request packet. The initiator gateway must also delay transmission at the end of the last byte of the request packet in order to ensure that packet aborts on the target bus are propagated back to the initiator bus. The initiator gateway must also ensure that a target device which responds just barely within the time-out limit on the target bus does not cause a time-out on the initiator bus, generally by asserting a delay on the initiator bus until this condition can be assured.

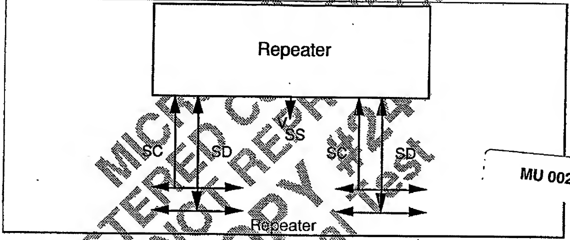
When a response packet is generated on the target bus (which may be from either the addressed target or some time-out generator), the packet is carried in the reverse direction by the gateway network. This response and any further packets

are carried until the end of the transaction. The contents of the response and further packets are not changed by the gateway network.

When a local Cerberus bus reset is received by a gateway, the reset is carried by the gateway network and each other gateway then re-transmits a reset transaction on all other local buses.

Repeater

A Cerberus bus may be extended by inserting repeaters. A repeater electrically separates two segments of a Cerberus bus, but provides a transparent linkage between these two segments. Using a repeater is advantageous when the capacitive load or clock skew between Cerberus devices on a large bus would require a reduction in the clock rate. The system designer must ensure that device addresses remain unique across what is logically a single serial bus.



Generally, a repeater will repeat each request packet seen on one side of the repeater on the other side, with a delay of at least one clock cycle. If two transactions appear nearly simultaneously on each side of the repeater, the repeater must abort one of the transactions and permit the other to be repeated. This arbitration must be performed fairly, such as by alternating which side of the repeater is preferred on consecutive collisions.

A simple repeater continues until the end of the transaction by repeating the response packets, which may appear on the same or opposite side as the original request packet of the transaction.

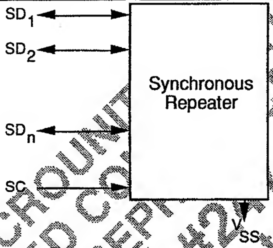
If the topology of the Cerberus is constructed so that only target devices exist on one side of the repeater, the design may be simplified by the elimination of the arbitration function. In such a case, transactions may only originate from the side designated to contain initiator-capable devices.

A more sophisticated repeater may "learn" which addresses are on each side of the repeater, and only repeat transactions which need to cross the repeater to be completed. Alternatively, a repeater may be constructed with knowledge of the

addresses to be placed on each side, such as addresses 0..127 on one side and addresses 128..255 on the other, again permitting the selective repeating of packets across the repeater.

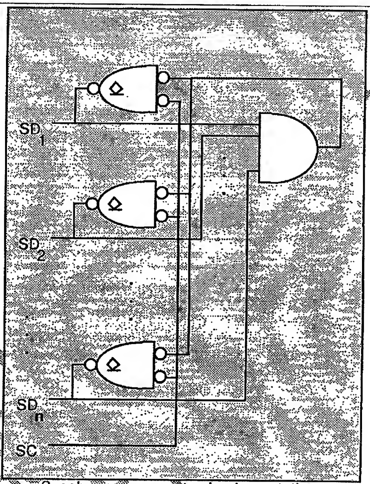
Synchronous Repeater

A very simple form of repeater may be employed to divide up the capacitive and leakage load on the SD signal of a Cerberus bus into two or more segments, when a common SC clock reference is used.



Synchronous repeater

The synchronous repeater samples each electrically-isolated segment of a logically-single Cerberus bus on the falling edge of each SC clock cycle, then broadcasts the logical AND of all the values on each segment during the SC clock low period.



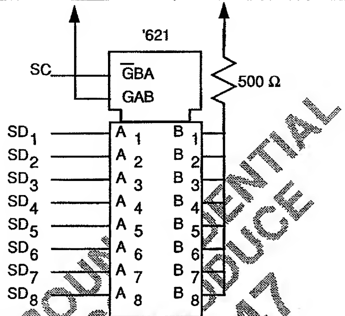
Synchronous repeater implementation

For large networks, this repeater improves performance by dividing up the RC delay by a factor of **n**, though two bus settling periods now occur on each SC clock period, so the speedup is approximately $\frac{n}{2}$.

MU 0023521

REDACTED

This circuit can be economically implemented using a single TTL '621 part and a pull-up resistor:



Eight-port synchronous repeater

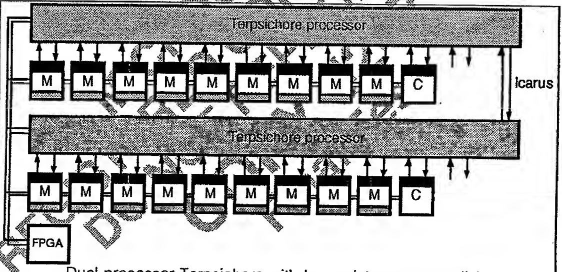
Icarus Interprocessor Protocol

MicroUnity's Icarus interprocessor protocol uses Hermes high-bandwidth channels to connect Terpsichore processors together, either directly or through external switching components, permitting the construction of shared-memory, coherently- or incoherently- cached multiprocessors. Icarus uses Hermes in the "Dual-Master Pair" configuration, and can be extended for use in "Multiple-Master Ring" configurations.

Internal daemons within Terpsichore perform and respond to Hermes write operations upon which the Icarus interprocessor communication protocol is embedded. These daemons provide for the generation of memory references to remote processors, for access to Terpsichore's local physical memory space, and for the transport of remote references to other remote processors.

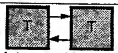
Interprocessor Topologies

The simplest multiprocessor configuration that can be built with the Icarus protocols is a dual-processor:



Dual-processor Terpsichore with Icarus interprocessor link

The diagram below represents the same dual-processor system, in a simpler notation:



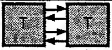
Dual-processor Terpsichore with Icarus interprocessor link

MU 0023523

In the configuration above, a pair of Hermes channels are connected together to form an Icarus Interprocessor link in the Dual-Master Pair configuration. A Cerberus bus connects all the system components together to facilitate system

configuration. The Terpsichore processors all run off of a common frequency clock, as required by the Hermes channels that connect between processors.

Dual Terpsichore processors with dual Icarus links may use both links to enhance system bandwidth:

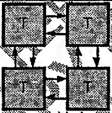


Dual-processor Terpsichore with Icarus interprocessor link

A Terpsichore processor's dual Icarus links, each in the Dual-Master Pair configuration may connect to two different processors. Using the Icarus Transponder daemons in each processor, several processors may be interconnected into a linear network of arbitrary size:

Four-processor Terpsichore with Icarus interprocessor links

The Icarus links may also join at the ends of the linear network, forming a ring or arbitrary size.

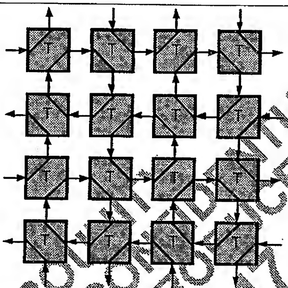


Four-processor Terpsichore with Icarus interprocessor links

In the configuration above, two Icarus links are connected to each Terpsichore processor, forming a single ring.

MU 0023524

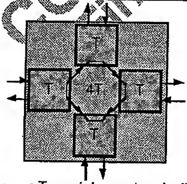
By connecting Icarus links into 4-master rings, providing Hermes' master forwarding for responses, using the Icarus Transponder daemons in each processor, processors may be interconnected into a two-dimensional network of arbitrary size:



Sixteen-processor Terpsichore with Icarus interprocessor links

In the configuration above, two Icarus links are connected to each Terpsichore processor, forming a single ring.

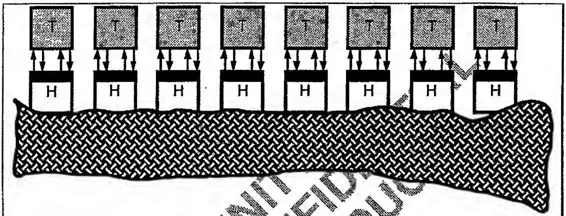
Other multidimensional topologies can be constructed by using multimaster rings as basic building blocks. An n-master ring ($n \leq 4$) of Terpsichore processors has n Icarus link-pairs available for connection into dual-master or multi-master configurations. For example, with a 4-master ring:



Four-processor Terpsichore ring building-block

These building blocks can then be assembled into radix-n switching networks:

By connecting Icarus links to external switching devices, multiprocessors with a large number of processors can be constructed with an arbitrary interconnection topology:



Eight-processor Terpsichore with Icarus links to switching fabric

In the configuration above, two Icarus links connect each Terpsichore processor to a switching fabric consisting of Hydra switches.

Link-level and Transaction-level Protocol

Icarus uses the Hermes protocol at the link level, and uses Hermes operations to embed a transaction-level protocol.

Two-packet link-action nomenclature

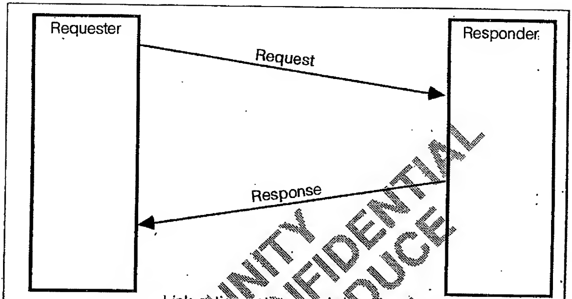
We designate the term "link-action" to describe the low-level packet protocol used between a Hermes master device and a Hermes slave device. The packets that make up a link-action contain a three-bit link-action identifier, or "lid," which permit up to eight outstanding link-actions to be in progress at any point in time.

Link-actions consist of two actions. Each packet transmitted on the Hermes ring corresponds to an action:

Request	the action taken by a requester to start the transaction.
Response	the action taken by the responder to finish the transaction.

Link-action nomenclature

These actions and their relation to the data flow is shown below:



Link-action actions and data flow

Four-packet transaction nomenclature

We designature the term "transaction" to describe the upper-level packet protocol used when embedding a four-packet, or "split" transaction above the link-level Hermes packet protocol.

Transactions are used when the latency of a transaction may require that more than eight actions are outstanding at a point in time, in order to maintain the desired throughput of the protocol. Embedding the transaction protocol above the link-action protocol limits the amount of link-level state which must be implemented.

Certain of the packets that make up a transaction contain an eight-bit transaction identifier, or "tid," which permit up to 256 outstanding transactions to be in progress at any point in time. These packets also contain link-action identifiers, lids, which connect these packets with others which are part of the transaction, but do not contain a tid.

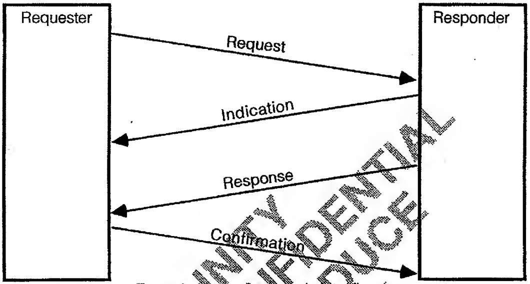
Transactions consist of four actions. Each action results in one or more link-level Hermes packets transmitted on the channels:

Request	the action taken by a requester to start the transaction.
Indication	the reception of a request by a responder.
Response	the action taken by the responder to finish the transaction.
Confirmation	the reception of the response by the requester.

Transaction nomenclature

MU 0023527

These actions and their relation to the data flow is shown below:



Transaction actions and data flow

The following table shows the relationship between transaction-level actions and link-level actions, showing typical transaction messages and link-action commands:

Transaction-level action	Typical transaction message	Link-level actions	Link-action command
Request	read/write-sizelet-request	Request	write-octlet
Indication	Remote-indication	Response	write-response
Response	read/write-sizelet-response	Request	write-octlet
Confirmation	Remote-confirmation	Response	write-response

Transaction protocol for Icarus Requester Daemon

MU 0023528

Highly Confidential

Icarus Action Format

Request and Response actions

A series of link-level write octlet operations comprise an Icarus request or response action. The address of the write operation contains target routing, transaction-id, commands and sequence information in the following format:

A remote request is a write octlet to an address of the form:

31 16	15 8	7 0
node	**tid**	**com**
16	8	6

with data of the form:

63   0
octlet
64

The **tid** field contains an 8-bit transaction id code which must be returned along with the remote response. The **tid** field value must be unique among all transactions originating from a node, but tid field values of transactions originating from distinct nodes may be equal.

The **com** field contains a 6-bit command code which, in the first octlet, designates the operation to be performed in a request action or the result returned in a response action. If the command code is in the range 0..31, in successive octlets, the value of the com field indicates whether the number of octlets to follow (0..9), such that the last octlet of a message contains a com field with a 0 value.

The **node** field contains a 16-bit node address which is the target of the action.

When embedded into a link-level write octlet operation, the Terpsichore requester daemon request appears on the Hermes in the form:

7		0
ma	2	lid
com		
tid		
$node_{7..0}$		
$node_{15..8}$		
$octlet_{63..56}$		
$octlet_{55..48}$		
$octlet_{47..40}$		
$octlet_{39..32}$		
$octlet_{31..24}$		
$octlet_{23..16}$		
$octlet_{15..8}$		
$octlet_{7..0}$		
check		

8

A transaction which has a payload of one octlet must use a link-level write octlet operation. A transaction which has a payload of greater than one octlet may successively use link-level write octlet operations to transmit the payload.

Indication and Confirmation actions

Indication and Confirmation actions consist of a series of link-level write octlet response packets, one for each octlet of the Request and Response actions.

Icarus Requester Daemon

When Terpsichore attempts a load or store to a physical address in which the high-order 16 bits are non-zero, the memory at that address is assumed to be present in the memory space of a remote Terpsichore processor. The Icarus Requester Daemon is an autonomous unit which attempts to satisfy such remote memory references by communicating with an external device, either another Terpsichore processor or a switching device which eventually reaches another Terpsichore processor.

These remote references are characterized by an eight-byte physical byte address, of which two bytes are used for specifying a processor node, and the remaining six bytes are used for specifying a local physical address on that processor node.

The Icarus Requester Daemon associates each remote memory reference with a transaction identifier[52] of eight bits, permitting up to 256 such remote references to be outstanding at any time; however, implementation limits within Terpsichore may set a smaller bound.

The Icarus Requester Daemon takes the role of the Transaction Requester, and an external device takes the role of the Transaction Responder. The daemon generates writes to a specified byte-channel and module address, which causes an external device to read or write remote octlets or cache lines in a remote memory. The daemon may have as many as two[53] link-level write requests outstanding at any point in time.

Terpsichore contains two such requester daemons which act concurrently to two different byte-channel and/or module addresses.

Icarus Responder Daemon

The Icarus Responder Daemon accepts writes from a specified byte-channel and module address, which enable an external device to generate transaction requests to read or write octlets or cache lines in the Terpsichore's local memory, or to generate Terpsichore events. The daemon also generates link-level writes to the same external device to communicate the responses to these transaction requests back to the external device.

Terpsichore contains two such responder daemons which act concurrently to two different byte-channel and/or module addresses.

An external device takes the role of the Transaction Requester, and the Icarus Responder takes the role of the Transaction Responder.

Icarus Transponder Daemon

The Icarus Transponder Daemon accepts writes from a specified Hermes channel and module address, which enable an external device to cause an Icarus Requester Daemon to generate a request on another Hermes channel and module address.

Terpsichore contains two such transponder daemons which act concurrently (back-to-back) between two different byte-channel and/or module addresses.

[52]The term "sequence number" is avoided here, because the transaction-tags are not necessarily sequential in nature.
[53]The number of link-level requests to be outstanding is still under study.

 REDACTED

MU 0023532

Icarus Request

The following table summarizes the commands used for Icarus requests and responses (response command shown in **bold**):

code	command	payload (octlets)
0	last octlet of multi-octlet command	
1..9	continuation octlet of multi-octlet command	
10..19	Reserved	
20	read incoherent strong cache-line	1
21	**read/add/swap octlet response**	**1**
22	read incoherent weak cache-line	1
23	**write response**	**1**
24	read allocate strong octlet	1
25	read noallocate strong octlet	1
26	read allocate weak octlet	1
27	read noallocate weak octlet	1
28	read allocate strong hexlet	1
29	read noallocate strong hexlet	1
30	read allocate weak hexlet	1
31	read noallocate weak hexlet	1
32	**read hexlet response**	**2**
33	**read incoherent cache-line response**	**8**
34	**read coherent cache-line response**	**9**
35..36	Reserved	
37	read coherent strong cache-line	2
38	Reserved	
39	read coherent weak cache-line	2
40..51	Reserved	
52	write coherent strong cache-line	10
53	write incoherent strong cache-line	9
54	write coherent weak cache-line	10
55	write incoherent weak cache-line	9
56	write allocate strong octlet	2
57	write noallocate strong octlet	2
58	write allocate weak octlet	2
59	write noallocate weak octlet	2
60	write allocate strong hexlet	3
61	write noallocate strong hexlet	3
62	write allocate weak hexlet	3
63	write noallocate weak hexlet	3
64	add-and-swap allocate strong octlet little-endian	2
65	add-and-swap noallocate strong octlet little-endian	2
66	add-and-swap allocate weak octlet little-endian	2
67	add-and-swap noallocate weak octlet little-endian	2

68..79	Reserved	
80	add-and-swap allocate strong octlet big-endian	2
81	add-and-swap noallocate strong octlet big-endian	2
82	add-and-swap allocate weak octlet big-endian	2
83	add-and-swap noallocate weak octlet big-endian	2
84	compare-and-swap allocate strong octlet	3
85	compare-and-swap noallocate strong octlet	3
86	compare-and-swap allocate weak octlet	3
87	compare-and-swap noallocate weak octlet	3
88	multiplex-and-swap allocate strong octlet	3
89	multiplex-and-swap noallocate strong octlet	3
90	multiplex-and-swap allocate weak octlet	3
91	multiplex-and-swap noallocate weak octlet	3
92	multiplex allocate strong octlet	3
93	multiplex noallocate strong octlet	3
94	multiplex allocate weak octlet	3
95	multiplex noallocate weak octlet	3
96-255	reserved	

Icarus Request commands

A remote {add,swap,or,and} octlet request is data of the form:

63 … 0
address
bytes 0..7
64

A remote read incoherent {strong,weak} cache-line request is data of the form:

63 … 0
address
64

A remote read coherent {strong,weak} cache-line request is data of the form:

63 … 0
address
coherence tag
64

MU 0023533

Highly Confidential

A remote write incoherent cache-line request is data of the form:

63 ... 0
address
bytes 0..7
bytes 8..15
bytes 16..23
bytes 24..31
bytes 32..39
bytes 40..47
bytes 48..55
bytes 56..63

64

A remote write coherent cache-line request is data of the form:

63 ... 0
address
coherence tag
bytes 0..7
bytes 8..15
bytes 16..23
bytes 24..31
bytes 32..39
bytes 40..47
bytes 48..55
bytes 56..63

64

A remote read {allocate,noallocate} {strong,weak} octlet request is data of the form:

63 ... 0
address

64

A remote write {allocate,noallocate} {strong,weak} octlet request is data of the form:

63 ... 0
address
bytes 0..7

64

MU 0023534

A remote read {allocate,noallocate} {strong,weak} hexlet request is data of the form:

63 ... 0
address
64

A remote write {allocate,noallocate} {strong,weak} hexlet request is data of the form:

63 ... 0
address
bytes 0..7
bytes 8..15
64

Icarus Indication

An Icarus Indication consists of a link-level write-response packet for each link-level write issued as an Icarus Request. Each link-level write-response packet contains the lid value of the link-level write-request packet. This serves both the link-level purpose of issuing a response and the ability to receive additional link-level requests and a transaction-level indication of receipt of the request and the ability to receive additional transaction-level requests.

Icarus Response

Icarus Responses consist of a series of one or more link-level write-octlet operations. The low-order bits of the addresses of the write operations contain commands and tid information, and the data is the contents read from memory.

The octlet stream contains transaction-level responses from the Terpsichore Responder daemon, which are summarized in the table below:

com	command	payload (octlets)
0	termination	
1..9	continuation	
10..22	Reserved	
23	write response	1
24..31	Reserved	
32	read/add/swap octlet response	2
33	read hexlet response	3
34	read incoherent cache-line response	9
35	read coherent cache-line response	10
7-255	reserved	

Icarus Response codes

MU 0023535

Highly Confidential

The **com** field contains an 8-bit message command, as given in the table previously.

The **tid** field contains the 8-bit transaction id code used in the request message.

The **node** field contains the 16-bit processor number used in the request message.

A remote {read,add,swap} octlet response is data of the form:

63	0
bytes 0..7	
64	

A remote read hexlet response is data of the form:

63	0
bytes 0..7	
bytes 8..15	
64	

A remote read incoherent cache-line response is data of the form:

63	0
bytes 0..7	
bytes 8..15	
bytes 16..23	
bytes 24..31	
bytes 32..39	
bytes 40..47	
bytes 48..55	
bytes 56..63	
64	

A remote write response is data of the form:

63	0
0	
64	

MU 0023536

A remote read coherent cache-line response is data of the form:

63 0

coherence tag
bytes 0..7
bytes 8..15
bytes 16..23
bytes 24..31
bytes 32..39
bytes 40..47
bytes 48..55
bytes 56..63

64

A remote write coherent cache-line response is data of the form:

63 0

coherence tag

64

Icarus Confirmation

An Icarus Confirmation consists of a link-level write-response packet for each link-level write issued as an Icarus Response. Each link-level write-response packet contains the lid value of the link-level write-request packet. This serves both the link-level purpose of issuing a response and indicating the ability to receive additional link-level requests and a transaction-level confirmation of receipt of the response and the ability to receive additional transaction-level requests.

Deadlock

The Icarus Requester, Responder, and Transponder daemons must act cooperatively to avoid deadlock that may arise due to an imbalance of requests in the system which prevent responses from being routed to their destination.

The requirements vary depending upon the characteristics of the system configuration, and the mechanisms for deadlock avoidance are still under study.

Principal mechanisms to employ are cycle-free-routing of requests, and the means to prioritize responses above requests in forwarding priority.

Error handling

The link-level packets contain a check byte which is designed to detect single-bit transmisstion errors in the Hermes channel.

When either party in an Icarus transaction receives a packet with a check error, it immediately shuts down input processing to avoid encountering further errors, as may arise from errors which disrupt the parsing of packets. It also generates an error packet, which ensures that the other party is notified of the error.

The target of an Icarus transaction must maintain a copy of the link-level address of the most recent correctly received link-level write operation in a Cerberus register. Terpsichore then will clear the error using the Cerberus channel, resetting the Hermes input processing. Each party then re-issues any outstanding link-level transactions.

The contents of the address field in the link-level protocol is used to ensure that the error handling mechanism does not result in missing or repeated operations. This is important, because unlike the link-level protocol, the transaction-level protocol contains non-idempotent operations.

```
        L.64            r3,-1(r8)
        G.MULADD.8      r4,r3,k10,r4
        L.64            r3,0(r8)
        G.MULADD.8      r4,r3,k11,r4
        L.64            r3,1(r8)
        G.MULADD.8      r4,r3,k12,r4
        A.ADD           r2,r8,row
        L.64            r3,-1(r2)
        G.MULADD.8      r4,r3,k20,r4
        L.64            r3,0(r2)
        G.MULADD.8      r4,r3,k21,r4
        L.64            r3,1(r2)
        G.MULADD.8      r4,r3,k22,r4
        G.COMPRESS.16   r4,r4,8
        S.64            r4,0(r9)
        A.ADD           r8,8
        A.ADD           r9,8
        B.NE            r8,r10,1b
```

With some obvious reordering of the address computation instructions, this can run in 10 cycles, assuming single-cycle latency for G.MULADD. Loop unrolling can be used to handle greater latency. The inner loop is 10 cycles per eight pixels, or 0.8 pixels/cycle. Counting each multiply as 8 operations and each multiply and add as 16 operations, we are running at 8+8*16=136 operations/loop / 10 cycles/loop = 13.6 operations/cycle.

Note that our design actually loads each pixel nine times, which is making good use of "excess" load bandwidth and data caching.

Filtering of Color Image

For a color image, we assume that the image is made up of pixels each 32 bits in size, 8 bits for each of red, green, blue, and alpha. We treat each component identically, so the same algorithm is used, but the offsets change slightly. A C version of the code is:

```
void ColorFilter(int8 *src, int8 *dst, int row, int pcount,
        int8 k00, int8 k01, int8 k02,
        int8 k10, int8 k11, int8 k12,
        int8 k20, int8 k21, int8 k22) {
    for (i=0; i!=4*pcount; i++) {
        dst[i] = (src[i-row-4]*k00 + src[i-row]*k01 + src[i-row+4]*k02 +
                 src[i-4]*k10 + src[i]*k11 + src[i+4]*k12 +
                 src[i+row-4]*k20 + src[i+row]*k21 + src[i+row+4]*k22)>>8;
    }
}
```

The assembler coding of the inner loop is:

```
1:      A.SUB           r2,r8,row
        L.64            r3,-4(r2)
        G.MUL.8         r4,r3,k00
        L.64            r3,0(r2)
        G.MULADD.8      r4,r3,k01,r4
        L.64            r3,4(r2)
        G.MULADD.8      r4,r3,k02,r4
        L.64            r3,-4(r8)
```

 REDACTED

```
        G.MULADD.8       r4,r3,k10,r4
        L.64             r3,0(r8)
        G.MULADD.8       r4,r3,k11,r4
        L.64             r3,4(r8)
        G.MULADD.8       r4,r3,k12,r4
        A.ADD            r2,r8,row
        L.64             r3,-4(r2)
        G.MULADD.8       r4,r3,k20,r4
        L.64             r3,0(r2)
        G.MULADD.8       r4,r3,k21,r4
        L.64             r3,4(r2)
        G.MULADD.8       r4,r3,k22,r4
        G.COMPRESS.128   r4,r4,8
        S.64             r4,0(r9)
        A.ADD            r8,8
        A.ADD            r9,8
        B.NE             r8,r10,1b
```

This uses the same algorithm as for the color image above. Operations are performed at the same rate, but since a pixel is represented by 32 bits, the pixel rate is four times slower. The inner loop runs at 10 cycles per 2 pixels, or 0.2 pixels/cycle.

Conversion of Monochrome to Color

To convert a monochrome image to a color image, we must triplicate each monochrome pixel level, into levels for red, green, and blue. The alpha level might be set to a constant level of 255, or merged in from a separate array.

```
void MonochromeToColor(int8 *src, int8 *dst, int pcount) {
    int i;

    for (i=0; i!=pcount; i++) {
        dst[i] = src[i];
        dst[4*i+1] = src[i];
        dst[4*i+2]= src[i];
        dst[4*i+3] = 255;
    }
}
```

Which results in the following inner loop (addressing operations and loop overhead omitted - they do not influence the operation count):

```
1:      L.64.B           r4,0(r8)
        G.SHUFFLE.16     r2,r4,r4
        G.SHUFFLE.16     r8,r4,r5        #r5 contains -1
        G.SHUFFLE.8      r6,r2,r8
        G.SHUFFLE.8      r8,r3,r9
        S.128.B          r6,0(r9)
        S.128.B          r8,16(r9)
        A.ADD            r8,8
        A.ADD            r9,32
        B.NE             r8,r10,1b
```

MU 0023540

The above sequence is 4 cycles per 8 pixels, or 2.0 pixels/cycle.

```
void MonochromeWithAlphaToColor(int8 *src, int8 *alpha, int8 *dst, int pcount) {
    int i;
```

Highly Confidential

```
    for (i=0; i!=pcount; i++) {
        dst[i] = src[i];
        dst[4*i+1] = src[i];
        dst[4*i+2]= src[i];
        dst[4*i+3] = alpha[i];
    }
}
```

Which results in the following inner loop:

```
1:      L.64.B          r4,0(r8)
        L.64.B          r5,0(r9)
        G.SHUFFLE.16    r2,r4,r4
        G.SHUFFLE.16    r8,r4,r5
        G.SHUFFLE.8     r6,r2,r8
        G.SHUFFLE.8     r8,r3,r9
        S.128.B         r6,0(r10)
        S.128.B         r8,16(r10)
        A.ADD           r8,8
        A.ADD           r9,8
        A.ADD           r10,32
        B.NE            r8,r11,1b
```

The above sequence is 4 cycles per 8 pixels, or 2.0 pixels/cycle.

Conversion of Color to Monochrome

To convert a color image to a monochrome image, a weighted sum of the red, green and blue components is generated. These weights, k0, k1, and k2, are selected so that k0+k1+k2 = 256, so overflow does not occur. The resulting weighted sum is truncated, rather than rounded, again, to avoid the possibility of overflow.

```
void ColorToMonochrome(int8 *src, int8 *dst, int pcount, int8 k0, int8 k1, int8 k2) {
    int i;

    for (i=0; i!=pcount; i++) {
        dst[i] = (src[4*i]*k0 + src[4*i+1]*k1 + src[4*i+2]*k2)>>8;
    }
}
```

Which results in the following inner loop:

```
        L.128.B         r2,0(r8)
        G.DEAL.16       r2,r2,r3        #k0k1...k0k1k200...k200
        L.128.B         r4,16(r8)
        G.DEAL.16       r6,r4,r5        #k0k1...k0k1k200...k200
        G.DEAL.8        r2,r2,r6        #k0k0...k0k0k1k1...k1k1
        G.DEAL.8        r4,r3,r7        #k2k2...k2k20000...0000
        G.MUL.8         r6,r2,k0
        G.MULADD.8      r6,r3,k1,r6
        G.MULADD.8      r6,r4,k2,r6
        G.COMPRESS.16   r6,r6,8         #toss away low precision
        S.64            r6,0(r9)
        A.ADD           r8,32
        A.ADD           r9,8
        B.NE            r8,r10,1b
```

MU 0023541

The above sequence is 8 cycles and writes 8 pixels, or 1.0 pixels/cycle.

The code below performs the same action, but also saves the alpha value into a second destination array.

```
void ColorToMonochrome(int8 *src, int8 *dst, int8 *alpha, int pcount,
        int8 k0, int8 k1, int8 k2) {
    int i;

    for (i=0, i!=pcount; i++) {
        dst[i] = (src[4*i]*k0 + src[4*i+1]*k1 + src[4*i+2]*k2)>>8;
        alpha[i] = src[4*i+3];
    }
}
```

Which results in the following inner loop:

```
        L.128           r2,0(r8)
        G.DEAL.16       r2,r2,r3        #k0k1...k0k1k200...k200
        L.128           r4,16(r8)
        G.DEAL.16       r6,r4,r5        #k0k1...k0k1k200...k200
        G.DEAL.8        r2,r2,r6        #k0k0...k0k0k1k1...k1k1
        G.DEAL.8        r4,r3,r7        #k2k2...k2k20000...0000
        S.64            r5,0(r10)
        G.MUL.8         r6,r2,k0
        G.MULADD.8      r6,r3,k1,r6
        G.MULADD.8      r6,r4,k2,r6
        G.COMPRESS.16   r6,r6,8         #toss away low precision
        S.64            r6,0(r9)
        A.ADD           r8,32
        A.ADD           r9,8
        A.ADD           r10,8
        B.NE            r8,r10,lb
```

The above sequence is 8 cycles and writes 8 pixels, or 1.0 pixels/cycle.

Image Warping

Image warping is the general process of selectively stretching and shrinking an image to make it appear to fit into a new shape, such as stretched around a sphere, or drawn on a surface that is tilted with respect to the viewing surface. A principal data structure used to generate such an effect is a set of decimated copies of the image, as shown in the diagram below. These are of particular value because interpolation of the elements of these copies produces a properly antialiased spatially-warped image. Note that the total size of this structure is always exactly four times larger than the original image. Each subarray is a copy of the image decimated in either the x or y direction, or both. The images get smaller and smaller going right and down in the array, until the image reaches a single dot. The original image need not be square or have sizes that are powers of two for this structure.

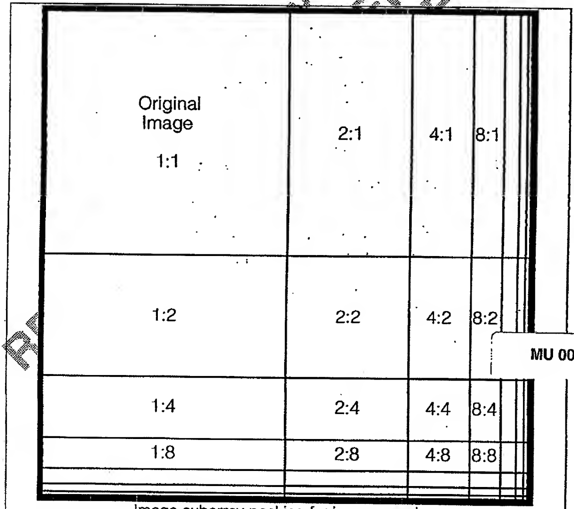


Image subarray packing for image warping

In the sections below, we explore two parts of the problem, the creation of the array containing this decimated image, and the antialiased selection of items in the table. These are the parts of the process which *must* be performed in real time for

real-time application of this process, the creation of the warping maps can often be precomputed, and are a function of the rendering system used.

Decimation of Monochrome Image

The process of generating the decimated images above can be divided into two parts, decimating in the horizontal direction only, and decimating in the vertical direction only. The former generates all the blocks to the right of the original image, and the latter generates the remaining blocks from those in the top row. This divides the problem into two parts, each using one-dimensional filtering, which is a great advantage because the amount of computation grows only linearly with the size of the filter function, rather than quadratically, when using two-dimensional filtering.

Our first example is the one-dimensional horizontal filter. We use a 5-point filter, specified by coefficients k0..k4 to specify the filter. These weights, k0..k4, are selected so that k0+k1+k2+k3+k4 = 256, so overflow does not occur. The resulting weighted sum is truncated, rather than rounded, again, to avoid the possibility of overflow.

```
void HorizontalDecimationMonochrome(int8 *src, int 8 *dst, int srow, int drow, int pcount,
        int8 k0, int8 k1, int8 k2, int8 k3, int8 k4) {
    int i,j,k;

    for (k=0,i=0; k!=pcount; ) {
        for (j=0; j!=drow; j++) {
            dst[k++] = (src[i-2]*k0 + src[i-1]*k1 + src[i]*k2 +
                    src[i+1]*k3 + src[i+2]*k4 )>>8;
            i+=2;
        }
        i+=srow-drow-drow;
    }
}
```

Which results in the following inner loop:

```
1:  L.128           r6,-2(r8)
    G.DEAL.8        r6,r6,r7
    G.MUL.8         r4,r6,k0
    G.MULADD.8      r4,r7,k1,r4
    L.128           r6,0(r8)
    G.DEAL.8.       r6,r6,r7
    G.MULADD.8      r4,r6,k2,r4
    G.MULADD.8      r4,r7,k3,r4
    L.128           r6,2(r8)
    G.DEAL.8        r6,r6,r7
    G.MULADD.8      r4,r6,k4,r4
    G.COMPRESS.16   r4,r4,8
    S.64            r4,0(r9)
    A.ADD           r8,16
    A.ADD           r9,8
    B.NE            r8,r10,1b
```

MU 0023544

This inner loop is 9 cycles per 8 pixels, or 0.9 Gpixels/sec, when the filter kernel size is 5 pixels wide. (For 3 pixels wide, the rate is 6 cycles per 8 pixels, or 1.3 pixels/cycle.)

Highly Confidential

When decimating in the vertical direction, the rate is even higher still:

```
void VerticalDecimationMonochrome(int8 *src, int 8 *dst, int srow, int drow, int pcount,
        int8 k0, int8 k1, int8 k2, int8 k3, int8 k4) {
    int i,j,k;

    for (k=0,i=0; k!=pcount; ) {
        for (j=0; j!=drow; j++) {
            dst[k++] = (src[i-2*srow]*k0 + src[i-srow]*k1 + src[i]*k2 +
                    src[i+srow]*k3 + src[i+2*srow]*k4 )>>8;
            i++;
        }
        i+=srow+srow-drow;
    }
}
```

Which results in the following inner loop:

```
1:      A.SUB           r2,r8,rowt2
        L.64            r3,0(r2)
        G.MUL.8         r4,r3,k0
        A.SUB           r2,r8,row
        L.64            r3,0(r2)
        G.MULADD.8      r4,r3,k1,r4
        L.64            r3,0(r8)
        G.MULADD.8      r4,r3,k2,r4
        A.ADD           r2,r8,row
        L.64            r3,0(r2)
        G.MULADD.8      r4,r3,k3,r4
        A.ADD           r2,r8,rowt2
        L.64            r3,0(r2)
        G.MULADD.8      r4,r3,k4,r4
        G.COMPRESS.16   r4,r4,8
        S.64            r4,0-8(r9)
        A.ADD           r8,8
        A.ADD           r9,8
        B.NE            r8,r10,1b
```

This runs in 6 cycles per 8 pixels, or 1.3 pixels/cycle. (For 3 pixels wide, the rate is 4 cycles per 8 pixels, or 2 pixels/cycle.)

To generate the decimated array shown above, for a n^2 image, n^2 pixels are generated in the horizontal direction, and $2n^2$ pixels are generated in the vertical direction. Using 5 pixel filter functions, this takes: $n^2/0.9 + 2n^2/1.3 = n^2*(1/0.9+2/1.3) = 2.63*n^2$ cycles. Thus, a 1024^2 image can be decimated in 2.8 Mcycles.

It is also possible to simultaneously decimate in the vertical and horizontal direction. While this may be more expensive that separately decimating in each direction, it permits the use of filter functions which do not factor into two parts. For this example, we assume a 2:1 decimation rate in each direction, and a 3x3 filter kernel. Real applications of decimation may use larger filter kernels, but this size serves to illustrate the techniques used. We assume here that pcount is a multiple of drow, and that drow<srow/2..

```
void DecimateMonochrome(int8 *src, int 8 *dst, int srow, int drow, int pcount,
        int8 k00, int8 k01, int8 k02,
```

```
        int8 k10, int8 k11, int8 k12,
        int8 k20, int8 k21, int8 k22) {
    int i,j,k;

    for (k=0,i=0; k!=pcount; ) {
        for (j=0; j!=drow; j++) {
            dst[k++] = (src[i-srow-1]*k00 + src[i-srow]*k01 + src[i-srow+1]*k02 +
                src[i-1]*k10 + src[i]*k11 + src[i+1]*k12 +
                src[i+srow-1]*k20 + src[i+srow]*k21 + src[i+srow+1]*k22)>>8;
            i+=2;
        }
        i+=2*(srow-drow);
    }
}
```

Assembler code for inner loop:

```
1:      A.SUB           r2,r8,srow
        L.128           r6,-1(r2)
        G.DEAL.8        r6,r6,r7
        G.MUL.8         r4,r6,k00
        G.MULADD.8      r4,r7,k01,r4
        L.128           r6,1(r2)
        G.DEAL.8        r6,r6,r7
        G.MULADD.8      r4,r6,k02,r4
        L.128           r6,-1(r8)
        G.DEAL.8        r6,r6,r7
        G.MULADD.8      r4,r6,k10,r4
        G.MULADD.8      r4,r7,k11,r4
        L.128           r6,1(r8)
        G.DEAL.8        r6,r6,r7
        G.MULADD.8      r4,r6,k12,r4
        A.ADD           r2,r8,srow
        L.128           r6,-1(r2)
        G.DEAL.8        r6,r6,r7
        G.MULADD.8      r4,r6,k00,r4
        G.MULADD.8      r4,r7,k01,r4
        L.128           r6,1(r2)
        G.DEAL.8        r6,r6,r7
        G.MULADD.8      r4,r6,k02,r4
        G.COMPRESS.16   r4,r4,8
        S.64            r4,0(r9)
        A.ADD           r8,16
        A.ADD           r9,8
        B.NE            r8,r10,1b
```


After some reordering of the address calculation instructions, the inner loop is 16 cycles per 8 pixels, or 0.5 pixels/cycle. Note that for 2:1 decimation in each direction, this is 4 times larger when expressed in terms of the input pixel rate: 2.0 pixels/cycle.

Because the filter function is an odd-number of pixels wide, 1/4 of the multiply bandwidth is effectively unused. For a 5x5 filter function, this would drop to 1/6 unused, and for an even number of pixels wide, none would be wasted. Compared to the two-dimensional filtering case, the multiplier bandwidth is less utilized because the index multiplier required the additional DEAL operations to be added.

Decimation of Color Image

Our first example is the one-dimensional horizontal filter. We use a 5-point filter, specified by coefficients k0..k4 to specify the filter. These weights, k0..k4, are selected so that k0+k1+k2+k3+k4 = 256, so overflow does not occur. The resulting weighted sum is truncated, rather than rounded, again, to avoid the possibility of overflow.

```
void HorizontalDecimationColor(int8 *src, int 8 *dst, int srow, int drow, int pcount,
        int8 k0, int8 k1, int8 k2, int8 k3, int8 k4) {
    int i,j,k;

    for (k=0,i=0; k!=pcount; ) {
        for (j=0; j!=drow; j++) {
            dst[k++] = (src[i-8]*k0 + src[i-4]*k1 + src[i]*k2 +
                    src[i+4]*k3 + src[i+8]*k4 )>>8;
            i++;
            dst[k++] = (src[i-8]*k0 + src[i-4]*k1 + src[i]*k2 +
                    src[i+4]*k3 + src[i+8]*k4 )>>8;
            i++;
            dst[k++] = (src[i-8]*k0 + src[i-4]*k1 + src[i]*k2 +
                    src[i+4]*k3 + src[i+8]*k4 )>>8;
            i++;
            dst[k++] = (src[i-8]*k0 + src[i-4]*k1 + src[i]*k2 +
                    src[i+4]*k3 + src[i+8]*k4 )>>8;
            i+=5;
        }
        i+=4*(srow+srow-drow-drow);
    }
}
```

Which results in the following inner loop:

```
1:      L.128           r6,-8(r8)
        G.DEAL.32       r6,r6,r7
        G.MUL.8         r4,r6,k0
        G.MULADD.8      r4,r7,k1,r4
        L.128           r6,0(r8)
        G.DEAL.32       r6,r6,r7
        G.MULADD.8      r4,r6,k2,r4
        G.MULADD.8      r4,r7,k3,r4
        L.128           r6,8(r8)
        G.DEAL.32       r6,r6,r7
        G.MULADD.8      r4,r6,k4,r4
        G.COMPRESS.16   r4,r4,8
        S.64            r4,0(r9)
        A.ADD           r8,16
        A.ADD           r9,8
        B.NE            r8,r10,1b
```

This inner loop is 9 cycles per 2 pixels, or 0.2 pixels/cycle, when the filter kernel size is 5 pixels wide. (For 3 pixels wide, the rate is 6 cycles per 2 pixels, or 0.3 pixels/cycle.)

When decimating in the vertical direction, the rate is even higher still:

```
void VerticalDecimationColor(int8 *src, int 8 *dst, int srow, int drow, int pcount,
        int8 k0, int8 k1, int8 k2, int8 k3, int8 k4) {
```

REDACTED

```
    int i,j,k;

    for (k=0,i=0; k!=pcount; ) {
        for (j=0; j!=4*drow; j++) {
            dst[k++] = (src[i-8*srow]*k0 + src[i-4*srow]*k1 + src[i]*k2 +
                      src[i+4*srow]*k3 + src[i+8*srow]*k4 )>>8;
            i++;
        }
        i+=4*(srow+srow-drow);
    }
}
```

Which results in the following inner loop:

```
1:      A.SUB           r2,r8,rowt8
        L.64            r3,0(r2)
        G.MUL.8         r4,r3,k0
        A.SUB           r2,r8,rowt4
        L.64            r3,0(r2)
        G.MULADD.8      r4,r3,k1,r4
        L.64            r3,0(r8)
        G.MULADD.8      r4,r3,k2,r4
        A.ADD           r2,r8,rowt4
        L.64            r3,0(r2)
        G.MULADD.8      r4,r3,k3,r4
        A.ADD           r2,r8,rowt8
        L.64            r3,0(r2)
        G.MULADD.8      r4,r3,k4,r4
        G.COMPRESS.16   r4,r4,8
        S.64            r4,0-8(r9)
        B.NE            r8,r10,1b
```

This runs in 6 cycles per 2 pixels, or 0.3 pixels/cycle. (For 3 pixels wide, the rate is 4 cycles per 2 pixels, or 0.5 pixels/cycle.)

To generate the decimated array shown above, for a n^2 image, n^2 pixels are generated in the horizontal direction, and $2n^2$ pixels are generated in the vertical direction. Using 5 pixel filter functions, this takes: $n^2/0.2 + 2n^2/0.3 = n^2*(1/0.2+2/0.3) = 10.5*n^2$ cycles. Thus, a 1024^2 image can be decimated in 11 Mcycles.

The last example in this section decimates a color signal in both directions simultaneously. We assume a 2:1 decimation rate in each direction, and a 3x3 filter kernel. Real applications of decimation may use larger filter kernels, but this size serves to illustrate the techniques used. We assume here that pcount is a multiple of drow, and that drow<srow/2..

```
void DecimateColor(int8 *src, int 8 *dst, int srow, int drow, int pcount,
        int8 k00, int8 k01, int8 k02,
        int8 k10, int8 k11, int8 k12,
        int8 k20, int8 k21, int8 k22) {
    int i,j,k;

    for (k=0,i=0; k!=4*pcount; ) {
        for (j=0; j!=drow; j++) {
            dst[k++]=(src[i-4*srow-4]*k00 + src[i-4*srow]*k01 + src[i-4*srow+4]*k02 +
                src[i-4]*k10 + src[i]*k11 + src[i+4]*k12 +
```

```
            src[i+4*srow-4]*k20 + src[i+4*srow]*k21 + src[i+4*srow+4]*k22)>>8;
        i++;
        dst[k++]=(src[i-4*srow-4]*k00 + src[i-4*srow]*k01 + src[i-4*srow+4]*k02 +
            src[i-4]*k10 + src[i]*k11 + src[i+4]*k12 +
            src[i+4*srow-4]*k20 + src[i+4*srow]*k21 + src[i+4*srow+4]*k22)>>8;
        i++;
        dst[k++]=(src[i-4*srow-4]*k00 + src[i-4*srow]*k01 + src[i-4*srow+4]*k02 +
            src[i-4]*k10 + src[i]*k11 + src[i+4]*k12 +
            src[i+4*srow-4]*k20 + src[i+4*srow]*k21 + src[i+4*srow+4]*k22)>>8;
        i++;
        dst[k++]=(src[i-4*srow-4]*k00 + src[i-4*srow]*k01 + src[i-4*srow+4]*k02 +
            src[i-4]*k10 + src[i]*k11 + src[i+4]*k12 +
            src[i+4*srow-4]*k20 + src[i+4*srow]*k21 + src[i+4*srow+4]*k22)>>8;
        i+=5;
      }
      i+=4*(srow+srow-drow-drow);
  }
}
```

Assembler code for inner loop:

```
1:      A.SUB           r2,r8,srow
        L.128           r6,-4(r2)
        G.DEAL.32       r6,r6
        G.MUL.8         r4,r6,k00
        G.MULADD.8      r4,r7,k01,r4
        L.128           r6,4(r2)
        G.DEAL.32       r6,r6
        G.MULADD.8      r4,r6,k02,r4
        L.128           r6,-4(r8)
        G.DEAL.32       r6,r6
        G.MULADD.8      r4,r6,k10,r4
        G.MULADD.8      r4,r7,k11,r4
        L.128           r6,4(r8)
        G.DEAL.32       r6,r6
        G.MULADD.8      r4,r6,k12,r4
        A.ADD           r2,r8,srow
        L.128           r6,-4(r2)
        G.DEAL.32       r6,r6
        G.MULADD.8      r4,r6,k00,r4
        G.MULADD.8      r4,r7,k01,r4
        L.128           r6,4(r2)
        G.DEAL.32       r6,r6
        G.MULADD.8      r4,r6,k02,r4
        G.COMPRESS.16   r4,r4,8
        S.64            r4,0(r9)
        A.ADD           r8,16
        A.ADD           r9,8
        B.NE            r8,r10,1b
```

After some reordering of the address calculation instructions, the inner loop is 16 cycles per 2 pixels, or 0.12 pixels/cycle.

Fractional Interpolation

This section is under construction.

Image Compression Applications

The following examples demonstrate key portions of JPEG and MPEG image compression applications. Both JPEG and MPEG applications rely on the use of a 2-dimensional Discrete Cosine Transform (DCT) to transform raster-image data into a frequency-based representation that is more amenable to entropy coding.

The following examples demonstrate several applications, listed below in summary form with the performance estimated. The estimates assume single-cycle loads and stores, that is, they do not account for losses due to cache misses. However, the memory reference patterns are very uniform, and with prefetching, they could be kept invisible.

Operation	cycles per pixel
Internal 8x8 Matrix Transpose	0.4
1-D Fixed-point 8-point Discrete Cosine Transform	1.0
2-D Fixed-point 8-by-8 Discrete Cosine Transform	2.8
1-D Floating-point 8-point Discrete Cosine Transform	0.6
2-D Floating-point 8-by-8 Discrete Cosine Transform	1.9
2-D Fixed-point 8-by-8 Discrete Cosine Transform for JPEG	2.3
2-D Floating-point 8-by-8 Discrete Cosine Transform for JPEG	1.4

Internal 8x8 Matrix Transpose

A 2-dimensional DCT can be performed on an 8-by-8 matrix of data by doing a series of 1-dimensional DCTs on each of the 8 rows of the matrix, and on each of the 8 columns of the matrix. A useful means to implement these operations is to perform a DCT on the rows (or columns) of the matrix, transpose the matrix, then perform a second, identical DCT, then transport the matrix again.

This example details the transposition of an 8-by-8 matrix of 16-bit values, stored consecutively in memory. The calculation is performed entirely in registers, using G.SHUFFLE instructions and a technique described in [54], in which the first and second halves of the matrix are shuffled $\log_2 N$ times.

Assume the matrix originally is in the order:

$$\begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 \\ 24 & 25 & 26 & 27 & 28 & 29 & 30 & 31 \\ 32 & 33 & 34 & 35 & 36 & 37 & 38 & 39 \\ 40 & 41 & 42 & 43 & 44 & 45 & 46 & 47 \\ 48 & 49 & 50 & 51 & 52 & 53 & 54 & 55 \\ 56 & 57 & 58 & 59 & 60 & 61 & 62 & 63 \end{bmatrix}$$

MU 0023550

[54] Stone, Harold, "Parallel Processing with the Perfect Shuffle," IEEE Transactions on Computers, Vol C-20, No. 2, February 1971, 153

After one shuffling, the matrix is in the order:

$$\begin{bmatrix} 0 & 32 & 1 & 33 & 2 & 34 & 3 & 35 \\ 4 & 36 & 5 & 37 & 6 & 38 & 7 & 39 \\ 8 & 40 & 9 & 41 & 10 & 42 & 11 & 43 \\ 12 & 44 & 13 & 45 & 14 & 46 & 15 & 47 \\ 16 & 48 & 17 & 49 & 18 & 50 & 19 & 51 \\ 20 & 52 & 21 & 53 & 22 & 54 & 23 & 55 \\ 24 & 56 & 25 & 27 & 26 & 58 & 27 & 59 \\ 28 & 60 & 29 & 61 & 30 & 62 & 31 & 63 \end{bmatrix}$$

After a second shuffling, the matrix is in the order:

$$\begin{bmatrix} 0 & 16 & 32 & 48 & 1 & 17 & 33 & 49 \\ 2 & 18 & 34 & 50 & 3 & 19 & 35 & 51 \\ 4 & 20 & 36 & 52 & 5 & 21 & 37 & 53 \\ 6 & 22 & 38 & 54 & 7 & 23 & 39 & 55 \\ 8 & 24 & 40 & 56 & 9 & 25 & 41 & 57 \\ 10 & 26 & 42 & 58 & 11 & 27 & 43 & 59 \\ 12 & 28 & 44 & 60 & 13 & 29 & 45 & 61 \\ 14 & 30 & 46 & 62 & 15 & 31 & 47 & 63 \end{bmatrix}$$

After a third shuffling, the matrix is in the order:

$$\begin{bmatrix} 0 & 8 & 16 & 24 & 32 & 40 & 48 & 56 \\ 1 & 9 & 17 & 25 & 33 & 41 & 49 & 57 \\ 2 & 10 & 18 & 26 & 34 & 42 & 50 & 58 \\ 3 & 11 & 19 & 27 & 35 & 43 & 51 & 59 \\ 4 & 12 & 20 & 28 & 36 & 44 & 52 & 60 \\ 5 & 13 & 21 & 29 & 37 & 45 & 53 & 61 \\ 6 & 14 & 22 & 30 & 38 & 46 & 54 & 62 \\ 7 & 15 & 23 & 31 & 39 & 47 & 55 & 63 \end{bmatrix}$$

C code for procedure:

```
void Matrix8By8Transpose(int16 *src, int16 *dst) {
    int16 tm0[64];
    int16 tm1[64];
    int i;

    for (i=0; i<32; i++) { tm0[2*i] = src[i]; tm0[2*i+1] = src[i+32]; }
    for (i=0; i<32; i++) { tm1[2*i] = tm0[i]; tm1[2*i+1] = tm0[i+32]; }
    for (i=0; i<32; i++) { dst[2*i] = tm1[i]; dst[2*i+1] = tm1[i+32]; }
}
```


Assembler code for procedure:

```
_Matrix8By8Transpose:
        L.128.I         r4,r2,0         # 00 01 02 03 04 05 06 07
        L.128.I         r12,r2,64       # 32 33 34 35 36 37 38 39
          G.SHUFFLE.8   r20,r4,r12      # 00 32 01 33 02 34 03 35
        L.128.I         r6,r2,16        # 08 09 10 11 12 13 14 15
          G.SHUFFLE.8   r22,r5,r13      # 04 36 05 37 06 38 07 39
        L.128.I         r14,r2,80       # 40 41 42 43 44 45 46 47
          G.SHUFFLE.8   r24,r6,r14      # 08 40 09 41 10 42 11 43
        L.128.I         r8,r2,32        # 16 17 18 19 20 21 22 23
```

REDACTED

```
    G.SHUFFLE.8    r26,r7,r15     # 12 44 13 45 14 46 15 47
  L.128.I          r16,r2,96      # 48 49 50 51 52 53 54 55
    G.SHUFFLE.8    r28,r8,r16     # 16 48 17 49 18 50 19 51
  L.128.I          r10,r2,48      # 24 25 26 27 28 29 30 31
    G.SHUFFLE.8    r30,r9,r17     # 20 52 21 53 22 54 23 55
  L.128.I          r18,r2,112     # 56 57 58 59 60 61 62 63
    G.SHUFFLE.8    r32,r10,r18    # 24 56 25 57 26 58 27 59
  G.SHUFFLE.8      r34,r11,r19    # 28 60 29 61 30 62 31 63

  G.SHUFFLE.8      r4,r20,r28     # 00 16 32 48 01 17 33 49
  G.SHUFFLE.8      r6,r21,r29     # 02 18 34 50 03 19 35 51
  G.SHUFFLE.8      r8,r22,r30     # 04 20 36 52 05 21 37 53
  G.SHUFFLE.8      r10,r23,r31    # 06 22 38 54 07 23 39 55
  G.SHUFFLE.8      r12,r24,r32    # 08 24 40 56 09 25 41 57
  G.SHUFFLE.8      r14,r25,r33    # 10 26 42 58 11 27 43 59
  G.SHUFFLE.8      r16,r26,r34    # 12 28 44 60 13 29 45 61
  G.SHUFFLE.8      r18,r27,r35    # 14 30 46 62 15 31 47 63

  G.SHUFFLE.8      r20,r4,r12     # 00 08 16 24 32 40 48 56
    S.128.I        r20,r3,0
  G.SHUFFLE.8      r22,r5,r13     # 01 09 17 25 33 41 49 57
    S.128.I        r22,r3,16
  G.SHUFFLE.8      r24,r6,r14     # 02 10 18 26 34 42 50 58
    S.128.I        r24,r3,32
  G.SHUFFLE.8      r26,r7,r15     # 03 11 19 27 35 43 51 59
    S.128.I        r26,r3,48
  G.SHUFFLE.8      r28,r8,r16     # 04 12 20 28 36 44 52 60
    S.128.I        r28,r3,64
  G.SHUFFLE.8      r30,r9,r17     # 05 13 21 29 37 45 53 61
    S.128.I        r30,r3,80
  G.SHUFFLE.8      r32,r10,r18    # 06 14 22 30 38 46 54 62
    S.128.I        r32,r3,96
  G.SHUFFLE.8      r34,r11,r19    # 07 15 23 31 39 47 55 63
    S.128.I        r34,r3,112
   B               r0
```

The resulting code transposes an 8-by-8 matrix using 25 cycles.

1-Dimensional Discrete Cosine Transform

The following code is based upon the Independent JPEG Group's software "jfwddct.c"[55], using 16-bit multiplies generating a 32-bit result.

```
#include "jinclude.h"

#define RIGHT_SHIFT(x,shft)  ((x) >> (shft))
#define LG2_DCT_SCALE 15  /* lose a little precision to avoid overflow */
#define ONE    ((INT32) 1)
#define DCT_SCALE (ONE << LG2_DCT_SCALE)

/* In some places we shift the inputs left by a couple more bits, */
/* so that they can be added to fractional results without too much */
/* loss of precision. */
#define LG2_OVERSCALE 2
#define OVERSCALE (ONE << LG2_OVERSCALE)
#define OVERSHIFT(x) ((x) <<= LG2_OVERSCALE)
```

MU 0023552

[55]Copyright (C) 1991, Thomas G. Lane.

```
/* Scale a fractional constant by DCT_SCALE */
#define FIX(x)  ((INT32) ((x) * DCT_SCALE + 0.5))

/* Scale a fractional constant by DCT_SCALE/OVERSCALE */
/* Such a constant can be multiplied with an overscaled input */
/* to produce something that's scaled by DCT_SCALE */
#define FIXO(x)  ((INT32) ((x) * DCT_SCALE / OVERSCALE + 0.5))

/* Descale and correctly round a value that's scaled by DCT_SCALE */
#define UNFIX(x)   RIGHT_SHIFT((x) + (ONE << (LG2_DCT_SCALE-1)), LG2_DCT_SCALE)

/* Same with an additional division by 2, ie, correctly rounded UNFIX(x/2) */
#define UNFIXH(x)  RIGHT_SHIFT((x) + (ONE << LG2_DCT_SCALE), LG2_DCT_SCALE+1)

/* Take a value scaled by DCT_SCALE and round to integer scaled by OVERSCALE */
#define UNFIXO(x)  RIGHT_SHIFT((x) + (ONE << (LG2_DCT_SCALE-1-LG2_OVERSCALE)),\
                   LG2_DCT_SCALE-LG2_OVERSCALE)

/* Here are the constants we need */
/* SIN_i_j is sine of i*pi/j, scaled by DCT_SCALE */
/* COS_i_j is cosine of i*pi/j, scaled by DCT_SCALE */

#define SIN_1_4 FIX(0.707106781)
#define COS_1_4 SIN_1_4

#define SIN_1_8 FIX(0.382683432)
#define COS_1_8 FIX(0.923879533)
#define SIN_3_8 COS_1_8
#define COS_3_8 SIN_1_8

#define SIN_1_16 FIX(0.195090322)
#define COS_1_16 FIX(0.980785280)
#define SIN_7_16 COS_1_16
#define COS_7_16 SIN_1_16

#define SIN_3_16 FIX(0.555570233)
#define COS_3_16 FIX(0.831469612)
#define SIN_5_16 COS_3_16
#define COS_5_16 SIN_3_16

/* OSIN_i_j is sine of i*pi/j, scaled by DCT_SCALE/OVERSCALE */
/* OCOS_i_j is cosine of i*pi/j, scaled by DCT_SCALE/OVERSCALE */

#define OSIN_1_4 FIXO(0.707106781)
#define OCOS_1_4 OSIN_1_4

#define OSIN_1_8 FIXO(0.382683432)
#define OCOS_1_8 FIXO(0.923879533)
#define OSIN_3_8 OCOS_1_8
#define OCOS_3_8 OSIN_1_8

#define OSIN_1_16 FIXO(0.195090322)
#define OCOS_1_16 FIXO(0.980785280)
#define OSIN_7_16 OCOS_1_16
#define OCOS_7_16 OSIN_1_16

#define OSIN_3_16 FIXO(0.555570233)
```

```
#define OCOS_3_16 FIXO(0.831469612)
#define OSIN_5_16 OCOS_3_16
#define OCOS_5_16 OSIN_3_16


/*
 * Perform a 1-dimensional DCT.
 * Note that this code is specialized to the case DCTSIZE = 8.
 */

INLINE
LOCAL void
fast_dct_8 (DCTELEM *in, int stride)
{
  /* many tmps have nonoverlapping lifetime -- flashy register colourers
   * should be able to do this lot very well
   */
  INT16 in0, in1, in2, in3, in4, in5, in6, in7;
  INT16 tmp0, tmp1, tmp2, tmp3, tmp4, tmp5, tmp6, tmp7;
  INT16 tmp10, tmp11, tmp12, tmp13;
  INT16 tmp14, tmp15, tmp16, tmp17;
  INT16 tmp25, tmp26;

  in0 = in[       0];
  in1 = in[stride  ];
  in2 = in[stride*2];
  in3 = in[stride*3];
  in4 = in[stride*4];
  in5 = in[stride*5];
  in6 = in[stride*6];
  in7 = in[stride*7];

  tmp0 = in7 + in0;
  tmp1 = in6 + in1;
  tmp2 = in5 + in2;
  tmp3 = in4 + in3;
  tmp4 = in3 - in4;
  tmp5 = in2 - in5;
  tmp6 = in1 - in6;
  tmp7 = in0 - in7;

  tmp10 = tmp3 + tmp0;
  tmp11 = tmp2 + tmp1;
  tmp12 = tmp1 - tmp2;
  tmp13 = tmp0 - tmp3;

  in[       0] = (DCTELEM) UNFIXH((tmp10 + tmp11) * SIN_1_4);
  in[stride*4] = (DCTELEM) UNFIXH((tmp10 - tmp11) * COS_1_4);

  in[stride*2] = (DCTELEM) UNFIXH(tmp13*COS_1_8 + tmp12*SIN_1_8);
  in[stride*6] = (DCTELEM) UNFIXH(tmp13*SIN_1_8 - tmp12*COS_1_8);

  tmp16 = UNFIXO((tmp6 + tmp5) * SIN_1_4);
  tmp15 = UNFIXO((tmp6 - tmp5) * COS_1_4);

  OVERSHIFT(tmp4);
  OVERSHIFT(tmp7);
```

```
  /* tmp4, tmp7, tmp15, tmp16 are overscaled by OVERSCALE */

  tmp14 = tmp4 + tmp15;
  tmp25 = tmp4 - tmp15;
  tmp26 = tmp7 - tmp16;
  tmp17 = tmp7 + tmp16;

  in[stride  ] = (DCTELEM) UNFIXH(tmp17*OCOS_1_16 + tmp14*OSIN_1_16);
  in[stride*7] = (DCTELEM) UNFIXH(tmp17*OCOS_7_16 - tmp14*OSIN_7_16);
  in[stride*5] = (DCTELEM) UNFIXH(tmp26*OCOS_5_16 + tmp25*OSIN_5_16);
  in[stride*3] = (DCTELEM) UNFIXH(tmp26*OCOS_3_16 - tmp25*OSIN_3_16);
}


/*
 * Perform the forward DCT on one block of samples.
 *
 * A 2-D DCT can be done by 1-D DCT on each row
 * followed by 1-D DCT on each column.
 */

GLOBAL void
j_fwd_dct (DCTBLOCK data)
{
  int i;

  for (i = 0; i < DCTSIZE; i++)
    fast_dct_8(data+i*DCTSIZE, 1);

  for (i = 0; i < DCTSIZE; i++)
    fast_dct_8(data+i, DCTSIZE);
}
```

The assembler code for the above procedure, called with stride=8, is as follows:

```
_fast_dct_8:
        L.128.I         r4,r2,0         #  in0 = in[       0];
        L.128.I         r6,r2,16        #  in1 = in[stride  ];
        L.128.I         r8,r2,32        #  in2 = in[stride*2];
        L.128.I         r10,r2,48       #  in3 = In[stride*3];
        L.128.I         r12,r2,64       #  in4 = in[stride*4];
        L.128.I         r14,r2,80       #  in5 = in[stride*5];
        L.128.I         r16,r2,96       #  in6 = in[stride*6];
        L.128.I         r18,r2,112      #  in7 = in[stride*7];
        G.ADD.16        r20,r18,r4      #  tmp0 = in7 + in0;
        G.ADD.16        r22,r16,r6      #  tmp1 = in6 + in1;
        G.ADD.16        r24,r14,r8      #  tmp2 = in5 + in2;
        G.ADD.16        r26,r12,r10     #  tmp3 = in4 + in3;
        G.SUB.16        r28,r10,r12     #  tmp4 = in3 - in4;
        G.SUB.16        r30,r8,r14      #  tmp5 = in2 - in5;
        G.SUB.16        r32,r6,r16      #  tmp6 = in1 - in6;
        G.SUB.16        r34,r4,r18      #  tmp7 = in0 - in7;
        G.ADD.16        r36,r26,r20     #  tmp10 = tmp3 + tmp0;
        G.ADD.16        r38,r24,r22     #  tmp11 = tmp2 + tmp1;
        G.SUB.16        r40,r22,r24     #  tmp12 = tmp1 - tmp2;
        G.SUB.16        r42,r20,r26     #  tmp13 = tmp0 - tmp3;
        G.ADD.16        r48,r36,r38     #  = tmp10 + tmp11
        G.MULADD.16     r44,r48,$SIN_1_4,$32768
        G.MULADD.16     r46,r49,$SIN_1_4,$32768
```

```
G.EXTRACT.I.16   r44,r44,r46,16
S.128.I          r44,r2,0        # in[    0] = ...
G.SUB.16         r48,r36,r38     # = tmp10 - tmp11
G.MULADD.16      r44,r48,$COS_1_4,$32768
G.MULADD.16      r46,r49,$COS_1_4,$32768
G.EXTRACT.I.16   r44,r44,r46,16
S.128.I          r44,r2,64       # in[stride*4] = ...
G.MULADD.16      r44,r42,$COS_1_8,$32768
G.MULADD.16      r46,r43,$COS_1_8,$32768
G.MULADD.16      r44,r40,$SIN_1_8,r44
G.MULADD.16      r46,r41,$SIN_1_8,r46
G.EXTRACT.I.16   r44,r44,r46,16
S.128.I          r44,r2,32       # in[stride*2] = ...
G.MULADD.16      r44,r42,$SIN_1_8,$32768
G.MULADD.16      r46,r43,$SIN_1_8,$32768
G.MULADD.16      r44,r40,$-COS_1_8,r44
G.MULADD.16      r46,r41,$-COS_1_8,r46
G.EXTRACT.I.16   r44,r44,r46,16
S.128.I          r44,r2,96       # in[stride*6] = ...
G.ADD.16         r48,r32,r30     # = tmp6 + tmp5
G.MULADD.16      r44,r48,$SIN_1_4,$4096
G.MULADD.16      r46,r49,$SIN_1_4,$4096
G.EXTRACT.I.16   r44,r44,r46,14  # tmp16 = ...
G.SUB.16         r48,r32,r30     # = tmp6 - tmp5
G.MULADD.16      r46,r48,$COS_1_4,$4096
G.MULADD.16      r48,r49,$COS_1_4,$4096
G.EXTRACT.I.16   r46,r46,r48,14  # tmp15 = ...
G.SHL.16         r28,r28,2       # OVERSHIFT(tmp4)
G.SHL.16         r34,r34,2       # OVERSHIFT(tmp7)
G.ADD.16         r48,r28,r46     # tmp14 = tmp4 + tmp15;
G.SUB.16         r50,r28,r46     # tmp25 = tmp4 - tmp15;
G.SUB.16         r52,r34,r44     # tmp26 = tmp7 - tmp16;
G.ADD.16         r54,r34,r44     # tmp17 = tmp7 + tmp16;
G.MULADD.16      r44,r54,$OCOS_1_16,$32768
G.MULADD.16      r46,r55,$OCOS_1_16,$32768
G.MULADD.16      r44,r48,$OSIN_1_16,r44
G.MULADD.16      r46,r49,$OSIN_1_16,r46
G.EXTRACT.I.16   r44,r44,r46,16
S.128.I          r44,r2,16       # in[stride] = ...
G.MULADD.16      r44,r54,$OCOS_7_16,$32768
G.MULADD.16      r46,r55,$OCOS_7_16,$32768
G.MULADD.16      r44,r48,$-OSIN_7_16,r44
G.MULADD.16      r46,r49,$-OSIN_7_16,r46
G.EXTRACT.I.16   r44,r44,r46,16
S.128.I          r44,r2,112      # in[stride*7] = ...
G.MULADD.16      r44,r52,$OCOS_5_16,$32768
G.MULADD.16      r46,r53,$OCOS_5_16,$32768
G.MULADD.16      r44,r50,$OSIN_5_16,r44
G.MULADD.16      r46,r51,$OSIN_5_16,r46
G.EXTRACT.I.16   r44,r44,r46,16
S.128.I          r44,r2,80       # in[stride*5] = ...
G.MULADD.16      r44,r52,$OCOS_3_16,$32768
G.MULADD.16      r46,r53,$OCOS_3_16,$32768
G.MULADD.16      r44,r50,$-OSIN_3_16,r44
G.MULADD.16      r46,r51,$-OSIN_3_16,r46
G.EXTRACT.I.16   r44,r44,r46,16
S.128.I          r44,r2,48       # in[stride*3] = ...
R                r0
```

The above code uses 10 G.ADD, 10 G.SUB, 32 G.MULADD, 10 G.EXTRACT.I, and 2 G.SHL instructions, which can be scheduled in 64 cycles. This code performs 8 1-dimensional DCTs at once, so it can be described as performing at 64/64 = 1.0 cycles/pixel.

2-Dimensional Discrete Cosine Transform

The code for a 2-dimensional DCT, uses the 1-dimensional DCT above, an 8x8 transform, a second 1-dimensional DCT, and a second 1-dimensional DCT. The load and store operations which are performed between these steps can be eliminated by procedure inlining, so we can estimate the performance by counting the Group instructions alone, which total to 2*64+2*24 or 176 cycles. The 2-dimensional DCT covers 64 pixels, which works out to a rate of 2.8 cycles/pixel. An inverse DCT should have similar performance characteristics.

Floating-point Discrete Cosine Transform

The DCT can also be performed using half precision (16-bit) floating-point operations. In this case, the accumulation of intermediate terms is performed using half-precision floating-point, so 50% of the G.MULADD instructions and 100% of the G.SHL and G.EXTRACT.I instructions can be removed. Also, 10 of the G.MULADD operations become simple G.MUL. Thus 8 1-Dimensional DCTs would use 10 GF.ADD, 10 GF.SUB, 10 GF.MUL, 3 GF.MULADD, 3 GF.MULSUB instructions, using 36 cycles, or 0.6 cycles/pixel, and the 2-dimensional 8x8 DCT uses 2*36+2*24 = 120 cycles, or 1.9 cycles/pixel. An inverse DCT should have similar performance characteristics.

Further enhancements when used in JPEG algorithm

Because the output of the DCT is scanned into a linear sequence of items, the final transpose operation can easily be eliminated. This reduces the fixed-point DCT cost to 2*64+24 = 152 cycles, or 2.4 cycles/pixel; the floating-point DCT cost is reducted to 2*36+24 = 96 cycles, or 1.5 cycles/pixel.

The following section demonstrates that the transpose cost can be reduced to 16 cycles, by using a combination of memory loads and stores and the G.SHUFFLE operations, producing a fixed-point DCT in 2*64 + 16 = 144 cycles, or 2.3 cycles/pixel and floating-point DCT in 2*36 + 16 = 88 cycles, or 1.4 cycles/pixel.

Other Matrix Applications

Internal 4x4 Matrix Transpose

This example details the transposition of a 4-by-4 matrix of 16-bit values, stored consecutively in memory. The calculation is performed entirely in registers, using

REDACTED

G.SHUFFLE instructions and a technique described in [56], in which the first and second halves of the matrix are shuffled $\log_2 N$ times.

Assume the matrix originally is in the order:

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 17 & 14 & 15 \end{bmatrix}$$

After one shuffling, the matrix is in the order:

$$\begin{bmatrix} 0 & 8 & 1 & 9 \\ 2 & 10 & 3 & 11 \\ 4 & 12 & 5 & 13 \\ 6 & 14 & 7 & 15 \end{bmatrix}$$

After a second shuffling, the matrix is in the order:

$$\begin{bmatrix} 0 & 4 & 8 & 12 \\ 1 & 5 & 9 & 13 \\ 2 & 6 & 10 & 14 \\ 3 & 7 & 11 & 15 \end{bmatrix}$$

C code for procedure:

```
void Matrix4By4Transpose(int16 *src, int16 *dst) {
    int16 tm0[16];
    int16 tm1[16];
    int i;

    for (i=0; i<8; i++) { tm0[2*i] = src[i]; tm0[2*i+1] = src[i+8]; }
    for (i=0; i<8; i++) { dst[2*i] = tm0[i]; dst[2*i+1] = tm0[i+8]; }
}
```

Assembler code for procedure:

```
_SubMatrixTranspose:
        L.128.I         r4,r2,0         # 00 01 02 03 04 05 06 07
        L.128.I         r6,r2,16        # 08 09 10 11 12 13 14 15
        G.SHUFFLE.8     r8,r4,r6        # 00 08 01 09 02 10 03 11
        G.SHUFFLE.8     r10,r5,r7       # 04 12 05 13 06 14 07 15

        G.SHUFFLE.8     r4,r8,r10       # 00 04 08 12 01 05 09 13
        S.128.I         r4,r3,0
        G.SHUFFLE.8     r6,r9,r11       # 02 06 10 14 03 07 11 15
        S.128.I         r6,r3,16
        B               r0
```

The resulting code transposes a 4-by-4 matrix using 5 cycles.

[56] Stone, Harold, "Parallel Processing with the Perfect Shuffle," IEEE Transactions on Computers, Vol C-20, No. 2, February 1971, 153

External Matrix Transpose

A large matrix may not fit in the register file all at once, and even if it could, the internal matrix transpose algorithm performs O(NlogN), as each doubling of the matrix size requires an additional shuffle.

To support the transpose of a large matrix, the internal matrix transpose algorithm can be extended to transpose individual blocks, or sub-matrices, of a large matrix, by modifying the code to specify the row size of of the matrix.[57]

If we consider each element in the left matrix below to be an 8-by-8 submatrix as above, the transpose of the matrix is the right matrix below, where each element of the right matrix is the transpose of the corresponding element in the left matrix. Note that elements 0, 9, 18, 27, 36, 45, 54, and 63 are transposed in-place, and that each of the other elements are transposed and exchanged with another element in the matrix. Thus another useful extension of the submatrix transpose algorithm transposes two submatrices simultaneously, writing them back in exchanged locations.[58]

$$
\begin{bmatrix}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 \\
24 & 25 & 26 & 27 & 28 & 29 & 30 & 31 \\
32 & 33 & 34 & 35 & 36 & 37 & 38 & 39 \\
40 & 41 & 42 & 43 & 44 & 45 & 46 & 47 \\
48 & 49 & 50 & 51 & 52 & 53 & 54 & 55 \\
56 & 57 & 58 & 59 & 60 & 61 & 62 & 63
\end{bmatrix}
\begin{bmatrix}
0 & 8 & 16 & 24 & 32 & 40 & 48 & 56 \\
1 & 9 & 17 & 25 & 33 & 41 & 49 & 57 \\
2 & 10 & 18 & 26 & 34 & 42 & 50 & 58 \\
3 & 11 & 19 & 27 & 35 & 43 & 51 & 59 \\
4 & 12 & 20 & 28 & 36 & 44 & 52 & 60 \\
5 & 13 & 21 & 29 & 37 & 45 & 53 & 61 \\
6 & 14 & 22 & 30 & 38 & 46 & 54 & 62 \\
7 & 15 & 23 & 31 & 39 & 47 & 55 & 63
\end{bmatrix}
$$

A preceding section describes how to transpose a 4x4 matrix, which can be easily extended to handle a 4x4 submatrix by splitting the L.128.I and S.128.I instructions each into pairs of L.64.I and S.64.I instructions. The cost of the 4x4 transpose is less than 25% of the cost of the 8x8 transpose, so an external matrix transpose using the 4x4 submatrix can be faster than using the 8x8 submatrix transpose.

Shaded Graphics Applications

This section is under construction.

MU 0023559

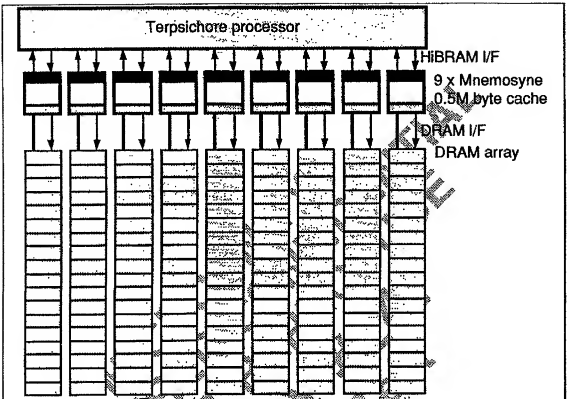
Mnemosyne System Application

MicroUnity's Terpsichore system architecture uses nine Mnemosyne memory devices in its base configuration, providing a nine byte-wide paths between the processor and memory. The memory devices are used to build a 0.5 Mbyte cache between Terpsichore's first level caches and DRAM-based main memory. The

[57]This modification uses A-type instructions to increment the src pointer by the row size between each L instructions, taking no additional cycles.

[58]For such a case, it is useful to use the indexed addressing form, so that the same index can be applied to the two pointers for the L and S instructions.

main memory store consists of 9, 18 or 36 banks of 1Mx72 arrays (each bank is eighteen 4 Mbit DRAMs), which yields 64,128 or 256 Mbytes of ECC memory with 8,16, or 32 Mbytes of directory storage.



Terpsichore memory system application

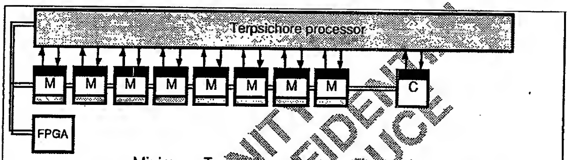
To further expand the DRAM memory and improve the bandwidth to memory, two or four Mnemosyne memory devices may be placed in each of the nine byte-wide paths. Such configurations use 18, 36, 72, or 144 banks of 1Mx72 arrays, which yields 128, 256, 512, or 1024 Mbytes of ECC memory with 16, 32, 64, or 128 Mbytes of directory storage.

Mnemosyne provide sufficient address bits to support up to 16Mx72 DRAM array banks, using as large as 64M bit DRAM parts when available. In such a configuration, memory sizes as large as 16 Gbytes of ECC memory can be constructed.

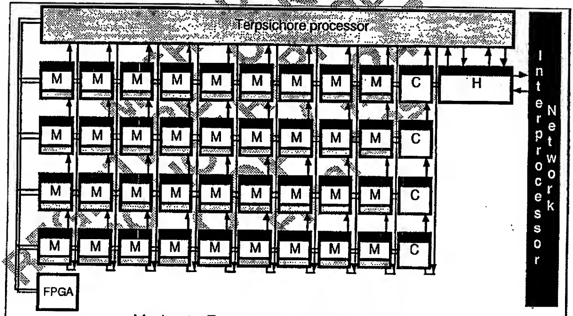
Terpsichore uses a 64-byte cache line size. Each cache line is associated with an octlet (8 bytes) of directory information, using one of the nine "Hermes channels" provided by a Mnemosyne device with its associated DRAM. The remaining eight of the nine Hermes channels contain the eight octlets (eight byte units) of the cache line data. In order to provide the means to access individual octlets of cache data and directory information at maximum bandwidth, the directory information is scattered evenly among eight of the nine byte lanes.

Typical Cerberus configurations

The number of devices in a typical Cerberus bus may vary from a minimum of about 11 devices (8 Mnemosyne, 1 Terpsichore, 1 Calliope, 1 FPGA), to a moderate amount of about 40 (36 Mnemosyne, 1 Terpsichore, 1 Calliope, 1 Hydra, 1 FPGA), or about 48 (36 Mnemosyne, 4 Terpsichore, 4 Calliope, 4 Hydra, 1 FPGA) to a maximum of about 157 devices (144 Mnemosyne, 4 Terpsichore, 4 Calliope, 4 Hydra, 1 FPGA).



Minimum Terpsichore system application

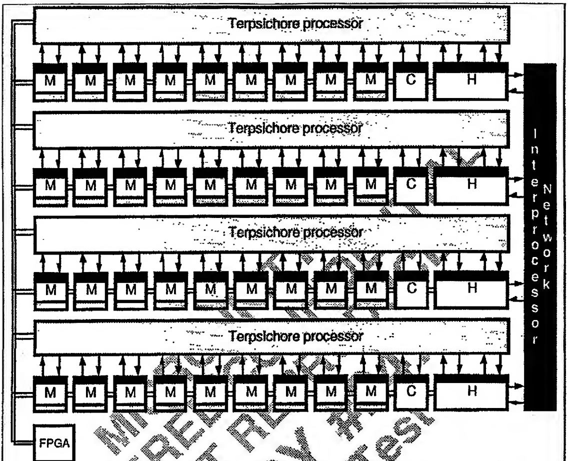


Moderate Terpsichore system application: one maximally-configured processor per board

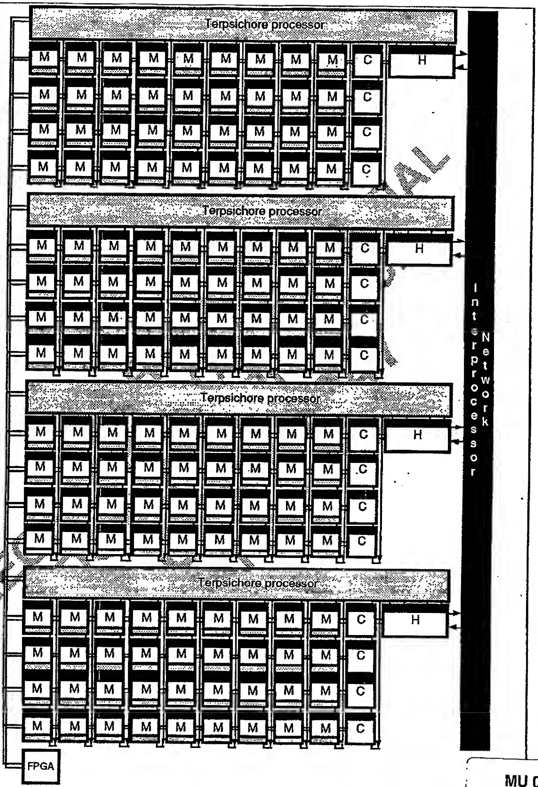
MU 0023561

Highly Confidential

REDACTED



Moderate Terpsichore system application
four minimally-configured processors per board



Maximum Terpsichore system application
four minimally-configured processors per board

Cerberus performance

When determining the performance of Cerberus, this 15:1 variation in the number of devices on the bus has a critical effect. The performance of these configurations with a resistive termination is estimated below:

	Constants				Basic	Full	Extended	
	ps/inch:			Terps	1	1	4	
	170			Mnemos	9	36	144	
	Pullup Rpu:			Calopes	1	2	4	
	500			Other	1	2	2	
	OC Rout (ohms)		Total devices		12	41	154	devices
	10		Bus length	2.5 "/device	30	102.5	385	inches
	Iin (uA):			3"/device	36	123	462	inches
	10	uA						
	OC falltime:		Tot. dev. cap.		48.00	164.00	616.00	pf
	5.00	ns						
	Dev. Cap (pf):		Line capacitance	100 ohms Zo	51.00	174.25	654.50	pf
	4	pf	at 2.5"/device	75 ohms Zo	68.00	232.33	872.67	pf
	Iol:							
	16	mA	Line capacitance	100 ohms Zo	61.20	209.10	785.40	pf
	p.u. volts		at 3"/device75 ohms	Zo	81.60	278.80	1,047.20	pf
	5	v						
	Tpd		System Cap.	2.5" 100 ohm	99.00	338.25	1,270.50	pf
	15	ns	device + line cap	2.5" 75 ohm	116.00	396.33	1,488.67	pf
	Tsetup			3" 100 ohm	109.20	373.10	1,401.40	pf
	15	ns		3" 75 ohm	129.60	442.80	1,663.20	pf
			Tau @ Rpu ohms	2.5" 100 ohm	49.50	169.13	635.25	ns
	Derived		Rpu * sys. cap.	2.5" 75 ohm	58.00	198.17	744.33	ns
	pf/inch:			3" 100 ohm	54.60	186.55	700.70	ns
100 ohm Zo	1.700			3" 75 ohm	64.80	221.40	831.60	ns
75 ohm Zo	2.267							
50 ohm Zo	3.400		Tau @ Iol	2.5" 100 ohm	30.94	105.70	397.03	ns
			sys. C * V /Iol	2.5" 75 ohm	36.25	123.85	465.21	ns
				3" 100 ohm	34.19	116.59	437.94	ns
				3" 75 ohm	40.50	138.38	519.75	ns
			Line delay	2.5"/dev	5.10	17.43	65.45	ns
			ps/in * line length	3"/dev	6.12	20.91	78.54	ns
			Max unterm Tr	2.5"/dev	20.40	69.70	261.80	ns
			4*line delay	3"/dev	24.48	83.64	314.16	ns
			Max unterm freq.	2.5"/dev	16.34	4.78	1.27	MHz
			1/(risetime*.003)	3"/dev	13.62	3.99	1.08	MHz
			Zterm for 1/2 dev.	100ohm/2.5"	49.02	14.35	3.82	Ohms
			cap. at OC fall time	100ohm/3"	40.85	11.96	3.18	Ohms
			DC input current		120	410	1540	uA
			Min Cycle freq	2.5" 100 ohm	11.15	4.27	1.28	MHz
			1000/Tau(Ih)+Tpd+Ts+	2.5" 75 ohm	10.18	3.80	1.10	MHz
			2*trace delay	3" 100 ohm	10.33	3.87	1.13	MHz
				3" 75 ohm	9.34	3.41	0.98	MHz

The use of a synchronous repeater, as described previously, would result in a significant performance increase in the Extended system.

MU 0023564

Index

MU 0023566

Highly Confidential

REDACTED

MU 0023572